ΤΕΙ Κρήτης
Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΘΕΜΑ: Ολοκληρωμένο σύστημα παροχής τουριστικών υπηρεσιών

ΣΠΟΥΔΑΣΤΡΙΑ: ΡΟΥΜΕΛΙΩΤΟΥ ΣΟΦΙΑ
ΑΜ: 3552

ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΜΗΧΑΝΟΛΟΓΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κο. Νικόλαο Παπαδάκη για την καθοδήγηση και την βοήθεια που μου πρόσφερε σε όλα τα στάδια της εκπόνησης της πτυχιακής μου εργασίας.

Ακόμα θα ήθελα να ευχαριστήσω την οικογένειά μου για την στήριξη που μου πρόσφερε όλο αυτό το διάστημα.

Abstract

The purpose of this thesis is to create a Tourist Guide for the city of Kavala. A user can be informed about hotels, rentals, sights etc. and also be able to book any of this for his vacations.

For this site i used for the front-end development html5, css3, sass, jquery and for the back-end development python and the Django framework. Other tools have been also used like gulp, bower and git as a version control .

Σύνοψη

Ο στόχος της αυτής της πτυχιακής εργασίας είναι η δημιουργία ενός τουριστικού οδηγού για την πόλη της Καβάλας. Ο χρήστης θα μπορεί να ενημερωθεί σχετικά με τα ξενοδοχεία της περιοχής, τα ενοικιαζόμενα που είναι διαθέσιμα καθώς και για τα αξιοθέατα και να πραγματοποιήσει κρατήσεις για την διεκπεραίωση των διακοπών του.

Για την υλοποίηση της σελίδας αυτής χρησιμοποίησα για το μπροστά κομμάτι (front-end development) html5, css3, sass, jquery και για το πίσω κομμάτι (back-end development) python και το Django framework. Για την διεκπεραίωση χρησιμοποιήθηκαν και άλλες τεχνολογίες όπως gulp, bower και git ως version control .

Περιεχόμενα

1. Περίληψη

Η πτυχιακή αυτή εργασία έχει ως στόχο την υλοποίηση μιας σελίδας για την πόλη της Καβάλας. Ένας οδηγός με εύκολη διαχείριση για τους επισκέπτες καθώς και για τους κατοίκους της πόλης. Παρέχει μια οργανωμένη εικόνα για τα ξενοδοχεία της περιοχής με φωτογραφικό υλικό, την κατηγορία που ανήκουν καθώς και την διαθεσιμότητά τους για τις ημερομηνίες ενδιαφέροντος του χρήστη. Η ίδια λογική υπάρχει και για τα καταστήματα ενοικίασης της περιοχής δίνοντας στον χρήστη μεγάλη ευελιξία επιλογών.

Κάθε χρήστης έχει τη δυνατότητα εγγραφής στη σελίδα και πρόσβαση στο δικό του προφίλ όπου υπάρχουν τα στοιχεία του, η δυνατότητα αποθήκευσης φωτογραφίας και προβολή του ιστορικού του με τις κρατήσεις που έχει διεκπεραιώσει στο παρελθόν.

Το περιεχόμενο της σελίδας είναι δυναμικό γεγονός που καθιστά τη συντήρηση της σελίδας εύκολη και οργανωμένη από τους διαχειριστές της. Όλες οι πληροφορίες βρίσκονται στο διαχειριστικό οπότε ανα πάσα στιγμή μπορεί να ανανεωθεί άμεσα με τις τελευταίες απαιτήσεις.

Η τεχνολογία που έχει χρησιμοποιηθεί μέσα από το Django fraimwork είναι πολύ ευέλικτη για την μελλοντική του παραμετροποίηση με βάση τις ανάγκες που μπορεί να προκύψουν από τους χρήστες της σελίδας.

1.1 Κίνητρο υλοποίησης της εργασίας

Ένας από τους λόγους δημιουργίας της εργασίας αυτής είναι η απόκτηση εμπειρίας πάνω στο κομμάτι του web development χρησιμοποιώντας νέα τεχνολογικά μέσα που θα με γεμίσουν εφόδια προς την ομαλή μετάβαση στην αγορά εργασίας.

1.2 Σκοπός και Στόχοι της εργασίας

Η χρήση του Internet γίνεται όλο και πιο καθοριστική στην καθημερινότητά μας γεγονός που καθιστά αναγκαία την δημιουργία σελίδων με σωστή δόμηση, φιλικές προς τον χρήστη και καλαίσθητες. Ακόμα η χρήση των κινητών για της ανάγκες μας είναι παγιωμένη και κατ επέκταση η δημιουργία responsive σελίδων που μπορούν να προσφέρουν την ίδια εμπειρία στον χρήστη ασχέτως της συσκευής που επιλέγει να χρησιμοποιεί. Στόχος, λοιπόν, είναι η δημιουργία μιας σελίδας η οποία μπορεί να καλύψει αυτές τις ανάγκες με τον καλύτερο δυνατό τρόπο.

2. Μεθοδολογία Υλοποίησης

Για τη δημιουργία της σελίδας χρησιμοποιήθηκε το Django MTV framwork (Model Template View) το οποίο είναι γραμμένο στη γλώσσα python. Για την παρουσίαση των δεδομένων στον χρήστη έχει χρησιμοποιηθεί HTML5, CSS3, Sass και JQuery.

Ακόμα για κάποιες λειτουργίες της σελίδας έχουν χρησιμοποιηθεί το Slick Slider το οποίο είναι ένα εργαλείο για τη δημιουργία Carousel φωτογραφιών καθώς και το Google Maps για την γεωγραφική αναπαράσταση των αντικειμένων που παρουσιάζουμε .

Για τη βάση δεδομένων έχει χρησιμοποιηθεί PostgreSQL που είναι ένα open source object-relational database σύστημα.

Ο κώδικας έχει γραφτεί στο Sublime το οποίο είναι ένας πάρα πολύ ευέλικτος text-editor με πολλές ευκολίες για την γρηγορότερη δημιουργία κώδικα.

Επιπλέον για το συντονισμό και την εξασφάλιση της σωστής δόμησης του κώδικα έχει χρησιμοποιηθεί το Git το οποίο είναι ένα version control system και το Github για την αποθήκευση του κώδικα online.

Το λειτουργικό σύστημα που χρησιμοποιήθηκε για την ολοκλήρωση της πτυχιακής εργασίας είναι linux Ubuntu, και οι οδηγίες εγκατάστασης είναι βασισμένες σε αυτό.

3. Σχέδιο Δράσης- Προγραμματιστικά εργαλεία

3. 1 Django

Το Django είναι ένα high-level Python Web framework και είναι ένα εργαλείο για να χτίζεις σελίδες και καλύπτει μεγάλη γκάμα αναγκών για ένα ολοκληρωμένο site. Παρέχει αυτόματα τη δημιουργία Admin για να διαχειριζόμαστε δυναμικά το περιεχόμενο της σελίδας καθώς και τους χρήστες της.

- Πολύ γρήγορο
    - Έχει σχεδιαστεί έτσι ώστε να βοηθάει τους προγραμματιστές ώστε να μπορούν να μεταβαίνουν από την ιδέα στην υλοποίηση όσο το δυνατόν γρηγορότερα.
- Πλήρως εξοπλισμένο
    - Το Django περιέχει δεκάδες extras τα οποία μπορούμε να χρησιμοποιήσουμε για να διαχειριστούμε συχνά Web development tasks. Ακόμα φροντίζει για το

user authentication, content administration, site maps, RSS feeds, και πολλά ακόμα tasks χωρίς καμία δυσκολία.

- Καθησυχαστικά ασφαλές
  - Το Django παίρνει την ασφάλεια πολύ σοβαρά και βοηθάει τους προγραμματιστές να αποφύγουν πολλά συχνά λάθη ασφαλείας όπως SQL injection, cross-site scripting, cross-site request forgery και clickjacking.
  - Το user authentication system παρέχει ένα ασφαλές τρόπο για να διαχειριστούμε τα user accounts και τα passwords.
- Εξαιρετικά επεκτάσιμη
  - Μερικά από τα πιο πολυσύχναστα sites χρησιμοποιούν τη δυνατότητα που έχει αυτό το framework να μπορεί γρήγορα και ευέλικτα να μεγαλώσει για να καλύψει τις απαιτήσεις του traffic.
- Απίστευτα πολύπλευρο
  - Εταιρίες, οργανισμοί και κυβερνήσεις έχουν χρησιμοποιήσει το Django για να χτίσουν διάφορες εφαρμογές από content management systems έως και social networks για επιστημονικές υπολογιστικές πλατφόρμες.

3.1.1 Εγκατάσταση του Django

Πριν μπορέσουμε να χρησιμοποιήσουμε το Django πρέπει να το εγκατστήσουμε.

- Εγκατάσταση της Python
  - Απο τη στιγμή που είναι ένα Python Web framework είναι αναγκαία η εγκατάσταση της.
  - Η Python περιλαμβάνει μια “ελαφριά” βάση την SQLite οπότε δεν χρειάζεται η εγκατάσταση βάσης αν δεν θέλουμε να χρησιμοποιήσουμε κάποια συγκεκριμένη πιο “βαριά” .
  - Για να επαληθεύσουμε ότι έχει εγκατασταθεί κανονικά πληκτρολογούμε python στο sell και περιμένουμε να μας επιστρέψει το ακόλουθο

```
Python 3.4.x
[GCC 4.x] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

- Στήσιμο της βάσης
  - Αυτό το βήμα μπορούμε να το προσπεράσουμε να θέλουμε να δουλέψουμε με την SQLite.
  - Για την διεκπεραίωση της πτυχιακής εργασίας έχει χρησιμοποιηθεί η PostgreSQL, χρειαζόμαστε το πακέτο psycopg2 το οποίο είναι postgreSQL adapter για την python.
- Εγκατάσταση του Django
  - Εγκατάσταση της τελευταίας έκδοσης χρησιμοποιώντας pip
    - Εγκαθηστούμε το pip στον υπολογιστή μας.

        - Σε αυτό το σημείο πρέπει να αναφέρουμε το virtualenv και το virtualenvwrapper. Αυτά τα εργαλεία μας παρέχουν απομονωμένα Python environments, τα οποία είναι πιο πρακτικό από το να εγκαθιστούμε πακέτα σε όλο το σύστημά μας. Ακόμα επιτρέπουν τη εγκατάσταση πακέτων χωρίς την άδεια διαχειριστή. Το contributing tutorial μας καθοδηγεί για το πως να δημιουργήσουμε ένα virtualenv στην Python 3.
        - Αφού έχουμε δημιουργήσει και ενεργοποιήσει το virtual environment, πληκτρολογούμε την εντολή pip install Django στο shell prompt.
- Επιβεβαίωση εγκατάστασης
    - Για να επιβεβαιώσουμε οτι η Python μπορεί να δει το Django, πληκτρολογούμε python στο shell. Μετά από το Python prompt, προσπαθούμε να κάνουμε import το Django:

```
>>> import django
>>> print(django.get_version())
1.10
```

- Είμαστε έτοιμοι για να ξεκινήσουμε.

3.1.2 Γράφοντας μια εφαρμογή

Από τη στιγμή που έχουμε χρησιμοποιήσει το virtualenv για να εγκαταστήσουμε το Django και την Python μέσα σε ένα απομονωμένο περιβάλλον πρέπει να μπούμε μέσα σε αυτό από το τερματικό μας, με το όνομα που του έχουμε δώσει, για να ξεκινήσουμε τη δημιουργία της εφαρμογής μας(στη προκειμένη περίπτωση):

- `$ workon django`

Μπορούμε να επιβεβαιώσουμε την εγκατάσταση του Django πληκτολογόντας την εντολή

- `$ python -m django --version`

3.1.3 Δημιουργώντας ένα project

Μέσα από το τερματικό πηγαίνουμε στο σημείο που θέλουμε να δημιουργήσουμε το project μας και πληκτρολογούμε

- `$ django-admin startproject kavala`

Αυτή η εντολή θα δημιουργήσει ένα kavala directory στο σημείο που βρισκόμαστε.

Ας δούμε τώρα τι δημιούργησε η εντολή startproject:

- ```
  kavala/
      manage.py
      kavala/
          __init__.py
          settings.py
          urls.py
          wsgi.py
  ```

Αυτά τα αρχεία είναι:

- Ο φάκελος kavala/ είναι ο root φάκελος του project σας. Το όνομα του δεν ενδιαφέρει το Django. Μπορείτε να το μετονομάσετε όπως θέλετε (είναι το όνομα που δώσατε στην εντολή django-admin startproject).

- manage.py: Αυτό το αρχείο αποτελεί εντολή (κονσόλας) την οποία μπορείτε να χρησιμοποιήσετε (και θα το κάνετε) για να αλληλεπιδράτε με το project σας για διάφορους λόγους (π.χ δημιουργία πινάκων στη database, εκκίνηση του προ-εγκατεστημένου server κλπ). Μπορείτε να διαβάσετε όλες τις λεπτομέρειες σχετικά με το αρχείο manage.py στο django-admin and manage.py.

- Ο εσωτερικός φάκελος kavala/ είναι ένα τυπικό Python package για το project σας. Το όνομα του (έχει σημασία) θα το χρησιμοποιήσετε για να κάνετε import οτιδήποτε σε άλλα Python packages (π.χ importkavala.urls).

- kavala/__init__.py: Ένα κενό αρχείο που λέει στην Python ότι αυτός ο φάκελος θα πρέπει να θεωρηθεί ως Python package. Αν είστε καινούργιος στην Python, διαβάστε περισσότερα σχετικά με τα packages στην επίσημη ιστοσελίδα της Python.

- kavala/settings.py: Ένα αρχείο που κρατά όλες τις ρυθμίσεις/παραμέτρους για αυτό το Django project (είναι η καρδιά ενός project). Το άρθρο Django settings θα σας εξηγήσει όλα όσα χρειάζεστε σχετικά με το πως δουλεύουν οι ρυθμίσεις αυτές.

- kavala/urls.py: Το αρχείο αυτό κρατά τους ορισμούς των URLs για αυτό το project. Με άλλα λόγια είναι ένας “πίνακας περιεχομένων” του Django site σας. Μπορείτε να διαβάστε περισσότερα για τα URLs στο άρθρο URL dispatcher.

- kavala/wsgi.py: Αυτό το αρχείο αποτελεί ένα σημείο εισόδου για τους Web servers οι οποίοι είναι συμβατοί με το WSGI specification. Δείτε περισσότερα στο άρθρο Πως να ανεβάσετε το site σας χρησιμοποιώντας το WSGI για περισσότερες λεπτομέρειες.

3.1.4 Ο development server
  ```

Τώρα που δημιουργήσαμε ένα Django project, θα πρέπει να σιγουρευτούμε ότι λειτουργεί. Από κονσόλα (γραμμή εντολών) μεταβείτε στο root φάκελο του project σας (kavala) και τρέξτε την ακόλουθη εντολή:

- `$ python manage.py runserver`

Θα δείτε την ακόλουθη έξοδο, στη κονσόλα:

- ```
  Performing system checks...

  System check identified no issues (0 silenced).

  You have unapplied migrations; your app may not work properly until
  they are applied.
  Run 'python manage.py migrate' to apply them.

  Φεβρουαρίου 13, 2017 - 15:50:53
  Django version 1.10, using settings 'kavala.settings'
  Starting development server at http://127.0.0.1:8000/
  Quit the server with CONTROL-C.
  ```

Μόλις εκκινήσατε τον Django development server, ένας ελαφρύς Web server γραμμένος αμιγώς σε Python. Συμπεριλάβαμε αυτή τη λειτουργία στο Django για να βλέπετε και να αναπτύσσετε το project σας γρήγορα, χωρίς να χρειαστεί να ασχολείστε με ρυθμίσεις του production server (όπως ο Apache), μέχρις ότου φθάσετε στο σημείο και ανεβάσετε το site σας στο internet.

Τώρα είναι ένα καλό σημείο να σημειωθεί το εξής: ποτέ μα ποτέ μην χρησιμοποιήσετε αυτό τον server (Django developemnt server) σε παραγωγικό περιβάλλον (production environment). Έχει σχεδιαστεί αποκλειστικά για αναπτυξιακούς λόγους (δηλαδή μόνο για να τρέχει τοπικά στον υπολογιστή σας).

Τώρα που ο server τρέχει, επισκεφτείτε τη διεύθυνση http://127.0.0.1:8000/ με τον αγαπημένο σας περιηγητή (firefox, chrome κλπ). Θα δείτε μια σελίδα με ένα μήνυμα καλωσορίσματος από το Django σε ανοιχτά μπλε χρώματα.
  ```

3.1.5 Δημιουργώντας το app

Τώρα που το περιβάλλον εργασίας σας – το “project” – είναι έτοιμο, μπορείτε να ξεκινήσετε να δημιουργείτε.

Κάθε application που γράφετε στο Django αποτελείται από ένα Python package (ένας φάκελος που περιέχει το αρχείο __init__.py) το οποίο πρέπει να ακολουθεί ορισμένους κανόνες.

Για άλλη μια φορά, το Django σας παρέχει την ανάλογη εντολή για να δημιουργήσετε αυτόματα τη δομή αυτού του φακέλου (Python package) ούτως ώστε να επικεντρωθείτε στο να γράψετε κώδικα παρά στο να δημιουργείτε φακέλους και υποφακέλους.

Στην εργασία αυτή έχουμε χρησιμοποιήσει την λογική των πολλαπλών apps για να έχουμε ένα καθαρό κώδικα και δόμηση των αρχείων. Ετσι η λογική για το κάθε app είναι απομονωμένη καθώς και όσα αρχεία σχετίζονται για το καθένα.

Για να δημιουργήσετε ένα app, σιγουρευτείτε ότι βρίσκεστε στο ίδιο επίπεδο με το αρχείο manage.py και πληκτρολογήστε την εντολή(για παράδειγμα το app hotels):

- `$ python manage.py startapp hotels`

Η εντολή θα δημιουργήσει ένα φάκελο hotels, ο οποίος αναπτύσσεται ως εξής:

- hotels/
  __init__.py
  admin.py
  apps.py
  migrations/
    __init__.py
  models.py
  tests.py
  views.py

3.2 PostgreSQL

3.2.1 Γενικές πληροφορίες

Η PostgreSQL είναι ένα πολύ δυνατό, ανοιχτού κώδικα object-relational σύστημα βάσεων. Έχει πάνω από 15 χρόνια ενεργής ανάπτυξης και αποδεδειγμένη αρχιτεκτονική κερδίζοντας δυνατή φήμη ως προς την αξιοπιστία, ακεραιότητα δεδομένων και ορθότητα.

Τρέχει σε όλα τα σημαντικά λειτουργικά συστήματα, περιλαμβάνοντας Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, macOS, Solaris, Tru64), και Windows. Είναι πλήρως ACID(atomic, consistent, isolated, durable) συμβατό, έχει πλήρη υποστήριξη για foreign keys, joins, views, triggers, και stored procedures(σε πολλαπλές γλώσσες).

Περιλαμβάνει κυρίως SQL:2008 data types, συμπεριλαμβάνοντας INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, και TIMESTAMP. Ακόμα υποστηρίζει την αποθήκευση δυαδικών μεγάλων αντικειμένων, συμπεριλαμβάνοντας εικόνες, ήχο ή βίντεο.

Διαθέτει εγγενείς διεπαφές προγραμματισμού για C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC μεταξύ άλλων, και εξερετικό documentation.

3.2.2 Εγκατάσταση της PostgreSQL

Για την εγκατάσταση της postgresql χρειάζεται να γράψουμε στο τερματικό τις εντολές

- `$ sudo apt-get update`
- `$ sudo apt-get install postgresql postgresql-contrib`

Με την εγκατάσταση δημιουργήθηκε ένας χρήστης με το όνομα postgres ο οποίος σχετίζεται με το default Postgres ρόλο. Για να μπορέσουμε να χρησιμοποιήσουμε την Postgres, μπορούμε να συνδεθούμε με αυτόν τον χρήστη.

Μπορούμε να μεταβούμε στον postgres λογαριασμό στον server μας πληκτρολογόντας

- `sudo -i -u postgres`

Μπορούμε τώρα να συνδεθούμε στη Postgres από το τερματικό πληκτρολογώντας

- `psql`

Είμαστε συνδεδεμένοι και έτοιμοι να αλληλεπιδράσουμε με το database management system άμεσα

Για να βγούμε απο το PostgreSQL τερματικό πληκτρολογούμε

- `postgres=#  \q`

Τώρα είμαστε πάλι στο Linux τερματικό μας.

3.2.3 Δημιουργία χρήστη

Για να δημιουργήσουμε έναν καινούργιο χρήστη πρέπει να γράψουμε την εντολή

- `$ sudo -u postgres createuser --interactive`

Με αυτή την εντολή εμφανίζονται κάποιες επιλογές και αναλόγως τις απαντήσεις εκτελούνται οι σωστές Postgres εντολές για τη δημιουργία του χρήστη ανάλογα με τις προδιαγραφές

- `Output`
    - `Enter name of role to add: kavala`
    - `Shall the new role be a superuser? (y/n) y`

3.2.4 Δημιουργία βάσης

Από προεπιλογή, μια άλλη υπόθεση που κάνει το σύστημα ελέγχου ταυτότητας Postgres είναι ότι θα υπάρξει μια βάση δεδομένων με το ίδιο όνομα όπως ο ρόλος που χρησιμοποιείται για να συνδεθείτε, την οποία ο ρόλος έχει πρόσβαση.

Έτσι, αν στην τελευταία ενότητα, δημιουργήσαμε ένα χρήστη που ονομάζεται kavala, αυτός ο ρόλος θα προσπαθήσει να συνδεθεί σε μια βάση δεδομένων η οποία καλείται επίσης kavala από προεπιλογή. Μπορείτε να δημιουργήσετε την κατάλληλη βάση δεδομένων με την εντολή createdb.

- `$ sudo -u postgres createdb kavala`

Για περισσότερες πληροφορίες επισκεφτείτε το DigitalOcean.

3.3 Django και Postgres

3.3.1 Documentation

Για να μάθουμε περισσότερα σχετικά με τη postgres μπορούμε να ανατρέξουμε στον οδηγό που υπάρχει στη σελίδα του Django.

3.3.2 Ρυθμίσεις για τη βάση στο project

Μέσα στο app που δημιουργήθηκε όταν φτιάξαμε το project μας (kavala) υπάρχει ένα αρχείο με το όνομα settings.py το οποίο κρατά όλες τις πληροφορίες σχετικά με την εφαρμογή μας. Εκεί βάζουμε τις ρυθμίσεις ώστε να μπορεί να συνδεθεί η εφαρμογή με τη βάση

settings.py

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'kavala',
        'USER': 'kavala',
        'PASSWORD': '123456',
        'HOST': '127.0.0.1',
        'PORT': '5432',
    }
}
```

3.4 Sass (Syntactically Awesome StyleSheets)

3.4.1 Γενικές πληροφορίες

Η Sass είναι μια επέκταση του CSS που προσθέτει δύναμη και κομψότητα με τη βασική γλώσσα. Μας επιτρέπει να χρησιμοποιήσουμε variables, nested rules, mixins, inline imports και πολλά άλλα, όλα με πλήρως CSS συμβατή σύνταξη. Η Sass βοηθά να κρατάμε μεγάλα stylesheets καλά οργανωμένα και τα μικρά stylesheets να τρέχουν γρήγορα.

Χαρακτηριστικά

- Πλήρος CSS-συμβατό
- Επέκταση γλώσσας όπως variables, nesting, and mixins
- Πολλές χρήσιμες functions για τη διαχείριση των χρωμάτων και άλλων τιμών
- Προηγμένα χαρακτηριστικά, όπως οι οδηγίες ελέγχου για τις βιβλιοθήκες
- Καλά διαμορφωμένη, προσαρμόσιμη έξοδο

3.4.2 Σύνταξη

Υπάρχουν δύο συντάξεις που διατίθενται για τη Sass. Η πρώτη, γνωστή ως SCSS (Sassy CSS), είναι μία προέκταση της σύνταξης του CSS. Αυτό σημαίνει ότι κάθε έγκυρο CSS stylesheet είναι ένα έγκυρο αρχείο SCSS με την ίδια έννοια. Επιπλέον, το SCSS καταλαβαίνει CSS hacks και vendor-specific σύνταξη, όπως η παλιά σύνταξη του φίλτρου του IE. Αυτή η σύνταξη είναι ενισχυμένη με τα χαρακτηριστικά του Sass που περιγράφονται παρακάτω. Αρχεία που χρησιμοποιούν την ακόλουθη σύνταξη έχουν την επέκταση .scss.

Το δεύτερο και με παλαιότερη σύνταξη, γνωστή ως indented σύνταξη (ή μερικές φορές απλά "Sass"), παρέχει ένα πιο συνοπτικό τρόπο γραφής CSS. Χρησιμοποιεί indented αντί για brackets για να δείξει το φώλιασμα των selectors, και νέες γραμμές αντί να μπαίνουν ερωτηματικά για το διαχωρισμό των properties. Μερικοί άνθρωποι βρίσκουν το Sass να είναι ευκολότερο να διαβαστεί και να γραφτεί πιο γρήγορα από ό, το SCSS. Η indented σύνταξη έχει τα ίδια χαρακτηριστικά, αν και ορισμένα από αυτά έχουν ελαφρώς διαφορετική σύνταξη. Αρχεία που χρησιμοποιούν την ακόλουθη σύνταξη έχουν την επέκταση .sass.

Για περισσότερες πληροφορίες επισκεφτείτε τη σελίδα του Sass.

3.4.3 Εγκατάσταση

Χρειάζεται να εγκαταστήσουμε πρώτα τη Ruby στον υπολογιστή μας. Μπορούμε να την εγκαταστήσουμε μέσα από το τερματικό με το apt package manager, rbenv ή rvm .

- `$ sudo su -c "gem install sass"`

Εγκατάσταση της Sass

- Ανοίγουμε το τερματικό.
- Η Ruby χρησιμοποιεί τα Gems για να διαχειριστεί τα διάφορα πακέτα κώδικα όπως η Sass. Στο τερματικό γράφουμε την εντολή
    - `$ gem install sass`
- Με αυτή την εντολή εγκαθιστούμε τη Sass και όσα dependencies χρειάζονται. Αν υπάρξει πρόβλημα με την εγκατάσταση τρέχουμε την εντολή ως sudo
    - `$ sudo gem install sass`
- Για να επιβεβαιώσουμε ότι έχει εγκατασταθεί σωστά πληκτρολογούμε
    - `sass-v`

Στην εργασία αυτή δεν ακολουθήθηκε αυτός ο τρόπος εγκατάστασης αλλά έγινε χρήση του gulp. Θα αναφερθούμε σε αυτό αμέσως μετά.

3.5 Gulp

3.5.1 Γενικά

Υπάρχουν πολλά tasks που εμπλέκονται με τη δημιουργία και την ανάπτυξη μιας front-end εφαρμογής. Κοινά tasks περιλαμβάνουν πράγματα όπως να παρακολουθούν τις αλλαγές στο αρχείο, συνένωση / ελαχιστοποίηση των αρχείων, να κάνουν prefixing στα αρχεία για διαφορετικούς browsers, και επισήμανση λαθών για την Javascript. Πριν από τους task runners όπως το Gulp και το Grunt, τα καθήκοντα αυτά συχνά γίνονταν με τη χρήση του shell ή bash scripts στη γραμμή εντολών.

Σε αυτή την εργασία έχουμε χρησιμοποιήσει το Gulp καθαρά για λόγους προτίμησις όσο αφορά το code styling.

Το Gulp είναι ένα node module άρα για να μπορέσουμε να το χρησιμοποιήσουμε πρέπει να εγκαταστήσουμε στον υπολογιστή μας την Node, κατά προτίμηση μέσω του nvm. Η Node έχει για package manager το npm.

Τα node modules εξαρτώνται από την έκδοση της node όπως επίσης και από το λειτουργικό σύστημα. Για αυτό το λόγο τα κάνουμε exlude από το version control. Το μέρος που μπορούμε να αποθηκεύσουμε τις πληροφορίες για τα πακέτα που χρησιμοποιεί η εφαρμογή μας , ώστε να μπορεί να τις βρει κάποιος που θα την κατεβάσει, είναι το αρχείο package.json. Αυτή η διαδικασία μπορεί να γίνει αυτόματα μέσω του npm.

Για παράδειγμα η εντολή

- `$ npm install gulp --save-dev`

Δημιουργεί τη παρακάτω γραμμή στο package.json

package.json

```
{
  "devDependencies": {
    "gulp": "^3.9.1",
  }
}
```

3.5.2 Plugins που χρησιμοποιήθηκαν

Για την υλοποίηση της σελίδας χρησιμοποιήθηκαν τα plugins

- gulp-sass για να μετατρέπεται ο κώδικας από sass σε css
    - `$ npm install gulp-sass --save-dev`

- gulp-autoprefixer για να μπαίνουν τα prefixes που χρειάζεται ο κάθε browser για να υποστηρίζεται και το css3
    - `$ npm install gulp-autoprefixer --save-dev`

- gulp-sourcemap για να μπορούμε να βλέπουμε το path των αρχείων που χρησιμοποιούμε
    - `$ npm install gulp-sourcemaps --save-dev`

Το αρχείο package.json έχει τη μορφή

package.json

```
{
  "devDependencies": {
    "gulp": "^3.9.1",
    "gulp-autoprefixer": "^3.1.1",
    "gulp-sass": "^2.3.2",
    "gulp-sourcemaps": "^1.6.0",
    "main-bower-files": "^2.13.1"
  }
}
```

3.5.3 Τα tasks

Ο κώδικας που ακολουθεί ενσωματώνει τα plugins που αναφέραμε πιο πάνω σε tasks τα οποία εκτελούνται αυτόματα με την εντολή “gulp” στο τερματικό μας .

Έχουμε δυο tasks το sass και το sass:watch τα οποία ενώνουμε στο task gulp

- Στο sass του λέμε τον φάκελο από τον οποίο να πάρει όλα τα αρχεία με την κατάληξη που θέλουμε, να τα περάσει μέσα από το sourcemap, αν υπάρχει κάποιο error να μας το εμφανίσει στο τερματικό, στη συνέχεια το περνάμε από το autoprefixer και τέλος σε πιο φάκελο να τα αποθηκεύσει.

- Το sass-watch παρακολουθεί ένα αρχείο και αν αλλάξει κάτι κάνει κάτι.

gulpfile.js

```
'use strict';

var gulp = require('gulp');
var sass = require('gulp-sass');
const autoprefixer = require('gulp-autoprefixer');
var sourcemaps = require('gulp-sourcemaps');

gulp.task('sass', function () {
  return gulp.src('./kavala/static/sass/**/*.sass')
      .pipe(sourcemaps.init())
    .pipe(sass().on('error', sass.logError))
    .pipe(autoprefixer({
            browsers: ['last 5 versions'],
            cascade: false
        }))
      .pipe(sourcemaps.write())
    .pipe(gulp.dest('./kavala/static/css'));
});

gulp.task('sass:watch', function () {
  gulp.watch('./kavala/static/sass/**/*.sass', ['sass']);
});

gulp.task('default', ['sass','sass:watch'])
```

3.6 Bower

3.6.1 Γενικά

Η παρακολούθηση όλων αυτών των πακέτων και η διασφάλιση ότι είναι ενημερωμένα (ή ότι να έχουμε τις ειδικές εκδόσεις που χρειαζόμαστε) είναι δύσκολο. Το Bower είναι ένα χρήσιμο εργαλείο που μας βοηθά σε αυτό το κομμάτι.

Το Bower μπορεί να διαχειριστεί τα components που περιέχουν HTML, CSS, JavaScript, γραμματοσειρές ή ακόμα και αρχεία εικόνας. Το Bower δεν κάνει concatenate ή minify του κώδικα ή οτιδήποτε άλλο - εγκαθιστά μόνο τις σωστές εκδόσεις των πακέτων που χρειαζόμαστε και τα dependencies τους.

Για να ξεκινήσετε, το Bower λειτουργεί με τη λήψη και την εγκατάσταση πακέτων από παντού, αναλαμβάνει το ψάξιμο, την εξεύρεση, το κατέβασμα και την αποθήκευση των πραγμάτων που ψάχνουμε. Το Bower παρακολουθεί αυτά τα πακέτα σε ένα αρχείο

διακήρυξης, το bower.json. Το πώς μπορούμε να χρησιμοποιήσουμε τα πακέτα είναι στο χέρι μας. Παρέχει hooks για να διευκολυνθεί η χρήση των πακέτων σε εργαλεία και στη ροή της εργασίας μας.

Έχει βελτιστοποιηθεί για το front-end. Αν πολλαπλά πακέτα εξαρτώνται από ένα πακέτο - η jQuery για παράδειγμα - το Bower θα κατεβάσει τη jQuery μόνο μία φορά. Αυτό είναι γνωστό ως μια επίπεδη γραφική παράσταση της εξάρτησης και βοηθά στη μείωση του φορτίου της σελίδας.

3.6.2 Εγκατάσταση του Bower και πακέτα

Το Bower είναι ένα command line utility, το εγκαθηστούμε με npm

- `$ npm install -g bower`

Χρειάζεται node,npm και το git(version control).

Στην εργασία αυτή για να έχουμε οργανωμένα τα πακέτα που εγκαθιστούμε έχουμε φτιάξει ένα αρχείο το .bowerrc μέσα στο οποίο του λέμε που να αποθηκεύει αυτά που κατεβάζει

.bowerrc
```
{
    "directory": "kavala/static/vendor"
}
```

Για παράδειγμα αν θέλουμε να βάλουμε την JQuery η εντολή στο τερματικό θα ήταν

- `$ bower install jquery --save`

Έχουμε χρησιμοποιήσει το bower για να εγκαταστήσουμε την jQuery, το slick-slider, το bootstrap και τη jquery-ui. Το αποτέλεσμα από αυτά τα πακέτα στο αρχείο bower.json είναι

bower.json
```
{
  "name": "kavala-tour",
  "dependencies": {
    "jquery": "^3.1.1",
    "slick-carousel": "^1.6.0",
    "jquery-ui": "^1.12.1",
    "bootstrap": "^3.3.7"
  }
}
```

3.7 Git (version control)

3.7.1 Γενικά

Το Git είναι ένα δωρεάν και open source σύστημα ελέγχου διανομής εκδόσεων έχει σχεδιαστεί για να χειρίζεται τα πάντα, από μικρά έως πολύ μεγάλα έργα με ταχύτητα και αποτελεσματικότητα.

Είναι εύκολο στη μάθηση και έχει ένα μικρό αποτύπωμα με αστραπιαία απόδοση. Υπερτερεί σε εργαλεία SCM όπως το Subversion, CVS, Perforce και ClearCase με χαρακτηριστικά όπως φθηνή τοπική διακλάδωση, βολικά staging areas και πολλαπλές ροές εργασίας.

Ουσιαστικά μας παρέχει τη δυνατότητα του πειραματισμού χωρίς να έχει “κόστος” για τον κώδικα που έχουμε ήδη γράψει. Αν θέλουμε να βάλουμε ένα καινούργιο feature στην εφαρμογή δημιουργούμε ένα καινούργιο branch και δουλεύουμε πάνω σε αυτό έχοντας κρατήσει στο main branch τον κωδικά μας στο προηγούμενο σημείο. Μπορούμε να ανατρέξουμε σε οποιοδήποτε commit έχουμε κάνει όσο παλιά και αν είναι αυτό και να δούμε τις διαφορές που έχει σε κάποιο σημείο.

Με τη δυνατότητα που μας παρέχει να κάνουμε συχνά commits μπορούμε να χωρίζουμε τα κομμάτια που δουλεύουμε ώστε να έχουμε καθαρή εικόνα και κυρίως καθαρά διαχωρισμένο κώδικα. Έτσι δίνοντας σωστούς τίτλους στα commits μας ξέρουμε που μπορούμε να ανατρέξουμε γρήγορα και εύκολα.

Ένα ακόμα σημαντικό κομμάτι που μας προσφέρει είναι η δυνατότητα να συνεργαζόμαστε με άλλους προγραμματιστές . Μπορούμε να δουλεύουμε ταυτόχρονα πάνω στο ίδιο project σε διαφορετικά branches ανεξάρτητα ο ένας από τον άλλον. Φυσικά όταν συμβαίνει αυτό υπάρχουν και κάποια βήματα που πρέπει να ακολουθούμε για την ομαλή συνένωση του κώδικα και για την αποφυγή επιπλοκών.

3.7.2 Χρήσιμες εντολές

Για να μετατρέψουμε ένα φάκελο σε git repository το μόνο που χρειάζεται είναι αφού έχουμε μπει στο σημείο που θέλουμε από το τερματικό να γράψουμε την εντολή

- `$ git init`

Τώρα ο φάκελός μας έχει μετατραπεί σε git repository

Για να δούμε σε ποιο brach είμαστε και ποιες αλλαγές έχουν γίνει στο working tree γράφουμε την εντολή

- `$ git status`

Για να προσθέσουμε στο staging area τις αλλαγές που έχουμε κάνει μέχρι εκείνο το σημείο γράφουμε την εντολή git add. Αν θέλουμε να προσθέσουμε όλα τα αρχεία που έχουμε πειράξει βάζουμε στο τέλος της εντολής μια τελεία(.) αλλιώς αν θέλουμε συγκεκριμένα αρχεία τα προσθέτουμε με το όνομά τους

- $ git add .

Για να καταγράψουμε τις staged αλλαγές στο repository γράφουμε την εντολή

- `$ git commit`

Η εντολή αυτή μας ανοίγει έναν editor για να γράψουμε τον τίτλο του commit ή και σχόλια για να είναι ευκολότερη η αναζήτηση αν τα χρειαστούμε.

Για να δούμε την λίστα όλων των commit που έχουμε κάνει γράφουμε την εντολή

- `$ git log`

Μια ακόμα εντολή που είναι πολύ χρήσιμη για την γρήγορη εύρεση κάποιου commit είναι

- `$ git log --grep=header`

Αυτή η εντολή θα επιστρέψει μόνο τα commits που περιέχουν την λέξη header.

Αν θέλουμε να δημιουργήσουμε ένα καινούργιο branch τότε γράφουμε την εντολή

- `$ git branch` *`branch-name`*

Για να αλλάξουμε το working tree στο καινούργιο branch γράφουμε την εντολή

- `$ git checkout` *`branch-name`*

Εναλλακτικά η παρακάτω εντολή δημιουργεί ένα καινούργιο branch και αλλάζει όλο το working tree να δείχνει σε αυτό

- `$ git checkout -b` *`branch-name`*

Αν θέλουμε να περάσουμε τις αλλαγές από ένα branch σε ένα άλλο , για παράδειγμα θέλουμε να περάσουμε τις αλλαγές στο master από το development γράφουμε

- `$ git checkout master`
- `$ git merge development`

Μέχρι τώρα δουλεύουμε τοπικά αν θέλουμε να έχουμε online τον κώδικά μας μπορούμε να χρησιμοποιήσουμε ένα git server ή μια hosting υπηρεσία κώδικα όπως το Github. Για να κάνουμε set το remote repository γράφουμε τις παρακάτω εντολές

- `$ git remote add origin` *`url-repository`*
- `$ git push -u origin master`

Κάθε φορά που θέλουμε να ανεβάσουμε τις αλλαγές online, αφού έχουμε ακολουθήσει τα προηγούμενα βήματα, γράφουμε την εντολή

- `$ git push`

Στην περίπτωση που δουλεύουμε με κάποιον άλλο ή έχουμε δουλέψει από άλλο υπολογιστή και οι αλλαγές δεν είναι αποθηκευμένες στον υπολογιστή μας, πριν ξεκινήσουμε να δουλεύουμε πρέπει να κατεβάσουμε τοπικά όλες τις αλλαγές με την εντολή

- `$ git pull`

Αυτές είναι κάποιες από τις βασικές εντολές του git. Υπάρχουν πάρα πολλές πληροφορίες και tutorials για τον χειρισμό του.

Πολλές φορές υπάρχουν αρχεία ή και ολόκληροι φάκελοι που δεν ανήκουν στο version control. Το μόνο που χρειάζεται για να κάνουμε exclude αυτά τα αρχεία από το git είναι να δημιουργήσουμε ένα αρχείο .gitignore στο root directory και να αναφέρουμε ποια είναι αυτά

.gitignore
```
node_modules
.vagrant

bower_components
kavala/static/vendor
```

4. Κύριο μέρος Πτυχιακής

4.1 Ανάλυση προβλήματος

Η υλοποίηση του Τουριστικού Οδηγού για την πόλη της Καβάλας, είχε ως σκοπό να ενημερώσει αλλά και να εξυπηρετήσει τους χρήστες της. Προσφέρει πληροφορίες για την πόλη αλλά και διάφορες άλλες υπηρεσίες. Μέσω των υπηρεσιών αυτών οι χρήστες μπορούν να οργανώσουν τις διακοπές τους σχετικά με τα αξιοθέατα της πόλης, τα ενοικιαζόμενα, τα ξενοδοχεία κτλ.

4.2 Δημιουργία βάσης

Το backbone όλης της εφαρμογής είναι η βάση, μια καλά σχεδιασμένη βάση σηματοδοτεί και την ευκολότερη υλοποίηση του κώδικα. Υπάρχουν κάποια βασικά βήματα που πρέπει να ακολουθούμε κατά τον σχεδιασμό μίας βάσης.

Το Database normalization, ή απλά normalization, είναι η διαδικασία της οργάνωσης των στηλών (χαρακτηριστικά) και των πινάκων (σχέσεις) μιας σχεσιακής βάσης δεδομένων για τη μείωση του πλεονασμού δεδομένων και τη βελτίωση της ακεραιότητας των δεδομένων.

4.2.1 Προβλήματα χωρίς Normalization

Χωρίς normalization, καθίσταται δύσκολο να χειριστεί και να ενημερωθεί η βάση δεδομένων, χωρίς να αντιμετωπίζει την απώλεια δεδομένων. Τα Insertion, Updation και Deletion Anamolies συμβαίνουν συχνά εάν η βάση δεδομένων δεν είναι normalized.

4.2.2 Normalization Rule

Οι κανόνες του normalization χωρίζονται στις ακόλουθες κανονικές μορφές

1. Πρώτη κανονική μορφή
2. Δεύτερη κανονική μορφή
3. Τρίτη κανονική μορφή

4.2.3 Πρώτη κανονική μορφή (1NF)

Σαν Πρώτη Κανονική Μορφή, δεν πρέπει υπάρχουν δύο γραμμές δεδομένων που να περιέχουν επαναλαμβανόμενη πληροφορία, δηλαδή κάθε σύνολο της στήλης πρέπει να έχει μια μοναδική τιμή, έτσι ώστε πολλαπλές στήλες δεν μπορούν να χρησιμοποιηθούν για να φέρουν την ίδια σειρά. Κάθε πίνακας πρέπει να οργανωμένος σε σειρές και κάθε σειρά θα πρέπει να έχει ένα πρωτεύον κλειδί που το διακρίνει ως μοναδικό.
Το πρωτεύον κλειδί είναι συνήθως μία μόνο στήλη, αλλά μερικές φορές περισσότερες από μία στήλη μπορούν να συνδυαστούν για να δημιουργήσουν ένα μοναδικό πρωτεύον κλειδί.

4.2.4 Δεύτερη κανονική μορφή (2NF)

Σύμφωνα με την δεύτερη κανονική μορφή δεν πρέπει να υπάρξει καμία εξάρτηση από καμία στήλη με το πρωτεύον κλειδί. Αυτό σημαίνει ότι για έναν πίνακα που έχει concatenated πρωτεύον κλειδί, κάθε στήλη του πίνακα που δεν είναι μέρος του πρωτεύοντος κλειδιού πρέπει να εξαρτάται από το σύνολο του concatenated κλειδιού για την ύπαρξή του. Εάν οποιαδήποτε στήλη εξαρτάται μόνο από το ένα μέρος του concatenated κλειδιού, τότε ο πίνακας αποτυγχάνει στη δεύτερη κανονική μορφή.

4.2.5 Τρίτη κανονική μορφή (3NF)

Η τρίτη κανονική μορφή υποδηλώνει ότι κάθε μη πρωτεύον χαρακτηριστικό του πίνακα πρέπει να εξαρτάται από το πρωτεύον κλειδί, ή μπορούμε να πούμε ότι, δεν πρέπει να υπάρχει η περίπτωση όπου ένα μη πρωτεύον χαρακτηριστικό καθορίζεται από ένα άλλο μη πρωτεύον χαρακτηριστικό . Έτσι, αυτή η μεταβατική λειτουργική εξάρτηση θα πρέπει να αφαιρεθεί από τον πίνακα και, επίσης, ο πίνακας πρέπει να είναι σε δεύτερη κανονική μορφή.

4.2.6 Σχεδιασμός βάσης

Η βάση μας είναι σχεδιασμένη στη τρίτη κανονική μορφή έκτος από το app του booking. Σε αυτή τη περίπτωση τα μοντέλα μας σχεδιάστηκαν με πολυμορφισμό για να καλύψουμε την ανάγκη της σύνδεσης μεταξύ μίας κράτησης που αφορά διαφορετικά αντικείμενα. Πιο συγκεκριμένα μία κράτηση μπορεί να περιέχει ένα δωμάτιο και ένα αυτοκίνητο.

Ο πολυμορφισμός είναι ένα χαρακτηριστικό των γλωσσών προγραμματισμού που επιτρέπει το χειρισμό τιμών διαφορετικών τύπων δεδομένων με χρήση μιας ομοιόμορφης διεπαφής. Η έννοια του παραμετρικού πολυμορφισμού εφαρμόζεται τόσο στους τύπους δεδομένων, όσο και στις συναρτήσεις. Μια συνάρτηση που μπορεί να αποτιμηθεί ή να εφαρμοστεί σε τιμές διαφορετικών τύπων είναι γνωστή ως πολυμορφική συνάρτηση. Ένας τύπος δεδομένων που εμφανίζεται ως γενικευμένου τύπου (π.χ. μια λίστα με στοιχεία οποιουδήποτε τύπου) ονομάζεται πολυμορφικός τύπος δεδομένων, ομοίως με το γενικευμένο τύπο από τον οποίο παράγονται οι εξειδικεύσεις.

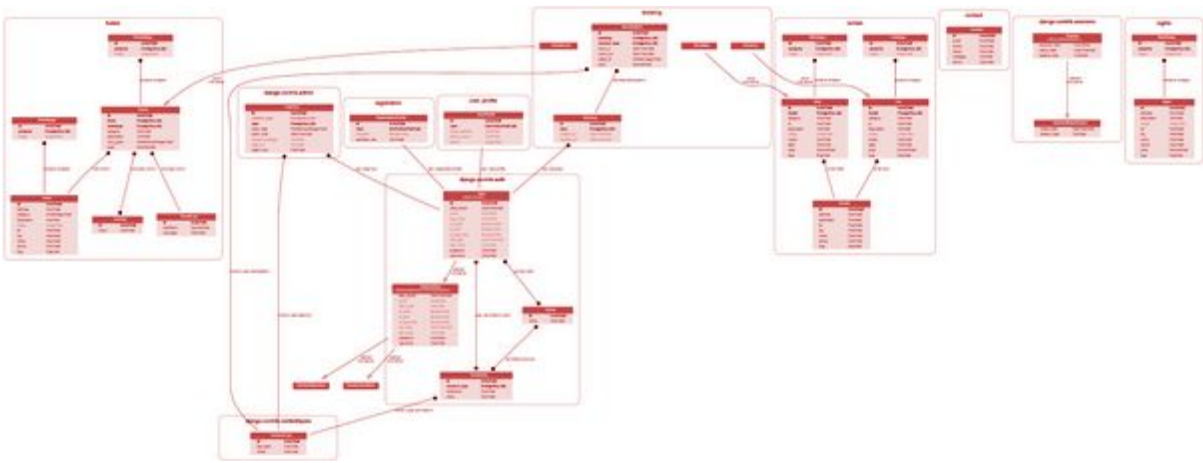
Έτσι για να δημιουργήσουμε την κλάση BookingItem χρησιμοποιήσαμε μέσα από το djano το contenttypes framework για τον πολυμορφισμό.

booking/models.py

```
class Booking(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now=False, auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True, auto_now_add=False)

    def __str__(self):
        return self.user.username


class BookingItem(models.Model):

    content_type = models.ForeignKey(ContentType,
on_delete=models.CASCADE)
    object_id = models.PositiveIntegerField()
    content_object = GenericForeignKey('content_type', 'object_id',
for_concrete_model=True)

    booking = models.ForeignKey(Booking, on_delete=models.CASCADE)
    check_in = models.DateTimeField(auto_now=False, auto_now_add=False)
    check_out = models.DateTimeField(auto_now=False, auto_now_add=False)
    price = models.DecimalField(max_digits=8, decimal_places=2)
```

Με αυτό τον τρόπο ένα BookingItem μπορεί να συνδεθεί με οποιοδήποτε άλλο μοντέλο.

Το σχεδιάγραμμα της βάσης είναι το ακόλουθο



1.Database graph

4.3 Τα app της εφαρμογής

Για την υλοποίηση της σελίδας χρησιμοποιήσαμε πολλαπλά app έτσι ώστε να υπάρχει οργάνωση στο κώδικα και να είναι εύκολη η διαχείριση τους όσο αφορά τα templates, τα models, url routing, views κ.α.

Η δομή της εφαρμογής είναι η ακόλουθη, θα δούμε κάποια από αυτά στη συνέχεια

- kavala
    - booking
    - contact
    - fixtures
    - hotels
    - kavala
    - media
    - node_modules
    - rentals
    - sights
    - user_profile

4.3.1 Django Admin

Το django μας παρέχει ένα πολύ δυνατό διαχειριστικό το οποίο είναι πάρα πολύ εύκολο να δημιουργηθεί . Το μόνο που χρειάζεται είναι να γράψουμε την εντολή

- `$ python manage.py createsuperuser`

Αφού τρέξουμε την εντολή μας ζητάει κάποιες πληροφορίες για να δημιουργήσουμε τον χρήστη μας.

- Username: admin
- Email address: admin@example.com
- Password: **********
  Password (again): *********
  Superuser created successfully.

Η σελίδα του admin βρίσκεται στη σελίδα http://127.0.0.1:8000/admin/

4.3.2 migrate-makemigrations

Η εντολή migrate κοιτάζει στη ρύθμιση INSTALLED_APPS και δημιουργεί τυχόν απαραίτητους πίνακες, βάση της ρύθμισης DATABASES του αρχείου kavala/settings.py και

βάση των database migrations τα οποία υπάρχουν σε κάθε app . Θα δείτε ένα μήνυμα για κάθε migration που ολοκληρώνεται .

Τα migrations αναπαριστούν στο Django, τις αλλαγές στα μοντέλα σας (και συνεπώς στο schema της βάσης δεδομένων σας) - δεν είναι τίποτε άλλο παρά αρχεία (Python κώδικας) στον υπολογιστή σας. Μπορείτε, αν θέλετε, να διαβάσετε το migration για τα καινούργια μοντέλα που δημιουργήσαμε. Βρίσκεται στο αρχείο hotels/migrations/0001_initial.py. Αλλά μην ανησυχείτε, δεν θα χρειαστεί να διαβάζετε αυτά τα αρχεία κάθε φορά που το Django τα δημιουργεί. Ωστόσο, είναι γραμμένα με φιλική-προς-τον-χρήστη μορφή ούτως ώστε να μπορείτε να τα επεξεργαστείτε στο μέλλον αν χρειαστεί.

Πρώτα τρέχουμε την εντολή

- `$ python manage.py makemigrations` *`app-name`*

Τρέχοντας την εντολή makemigrations, λέτε στο Django ότι έχετε κάνει κάποιες αλλαγές στο μοντέλο σας (σε αυτή την περίπτωση, έχετε δημιουργήσει καινούργια μοντέλα) και ότι θα θέλατε αυτές οι αλλαγές να αποθηκευτούν ως migration.

Στη συνέχεια πληκτρολογούμε την εντολή

- `$ python manage.py migrate`

4.3.3 App hotels models και admin

Ξεκινώντας πρέπει να δημιουργήσουμε τα μοντέλα μας για τη βάση, πώς θα αποθηκεύονται οι πληροφορίες μας και ποιες θα είναι οι σχέσεις μεταξύ τους.

Αρχικά φτιάξαμε τη κλάση Hotel μέσα στην οποία αποθηκεύουμε τις βασικές πληροφορίες που σχετίζονται με ένα ξενοδοχείο όπως όνομα, τηλέφωνο, διεύθυνση, περιγραφή, κατηγορία, εικόνα γεωγραφικές συντεταγμένες .

Για να μπορέσουμε να έχουμε πολλές φωτογραφίες για ένα ξενοδοχείο, δημιουργήσαμε την κλάση HotelImage η οποία με τη χρήση ξένων κλειδιών μας δίνει το επιθυμητό αποτέλεσμα. Ο κώδικας που το πετυχαίνουμε αυτό είναι ο ακόλουθος

hotels/models.py
```
property = models.ForeignKey(Hotel, related_name='images')
image = models.ImageField(upload_to = 'hotel_image', blank=True)
```

Οι εικόνες αποθηκεύονται μέσα στο φάκελο hotel_image στο app media. Για να γίνει αυτό χρειάστηκαν κάποιες ρυθμίσεις όπως οι παρακάτω στο αρχείο settings.py

settings.py

```
MEDIA_DIR =  os.path.join(BASE_DIR, 'media')
MEDIA_ROOT = MEDIA_DIR
MEDIA_URL = '/media/'
```

Δημιουργήσαμε την κλάση RoomType η οποία κρατά πληροφορία σχετικά με το είδος των δωματίων που υπάρχουν και το ποσοστό % για τη τιμή ανάλογα το είδος. Η σύνδεσή τους με τα δωμάτια γίνεται μέσα από την κλάση Room, όπου εκτός από τις πληροφορίες που κρατά σχετικά με τα δωμάτια, κρατά και την πληροφορία για τον τύπο του δωματίου καθώς και για το ξενοδοχείο στο οποίο ανήκει. Οι εντολές που το υλοποιούν αυτό είναι

hotels/models.py

```
hotel = models.ForeignKey(Hotel, on_delete=models.CASCADE)
roomtype = models.ForeignKey(RoomType, on_delete=models.CASCADE)
```

Για τις φωτογραφίες των δωματίων φτιάξαμε την κλάση RoomImage η οποία λειτουργεί με την ίδια λογική των ξενοδοχείων.

hotels/models.py

```
property = models.ForeignKey(Room, related_name='images')
image = models.ImageField(upload_to = 'room_image', blank=True)
```

Η κλάση Amenity κρατά πληροφορίες σχετικά με τις παροχές που έχει το κάθε ξενοδοχείο.

Πριν συνεχίσουμε σε αυτό το σημείο πρέπει να δηλώσουμε μέσα στο αρχείο settings.py στη λίστα των installed apps την εφαρμογή μας hotels για να ξέρει το project μας πως την χρησιμοποιούμε.

```
INSTALLED_APPS = [
      .
      .

    'hotels',
]
```

Για να μπορέσουμε να δούμε και να διαχειριστούμε τα μοντέλα μας στο διαχειριστικό πρέπει να τα δηλώσουμε μέσα στο αρχείο admin.py το οποίο βρίσκετε μέσα στο app hotels. Αν θέλουμε απλά να παρουσιάσουμε ένα μοντέλο τότε απλά το δηλώνουμε με τον παρακάτω τρόπο

hotels/admin.py

```
admin.site.register(Amenity)
```

Αν όμως θέλουμε να δημιουργήσουμε πιο περίπλοκες σχέσεις πρέπει να ανατρέξουμε στο documentation για να βρούμε τι είναι αυτό που θέλουμε να πετύχουμε. Για παράδειγμα στο μοντέλο Hotel θέλουμε να εμφανίζουμε και τα δωμάτια που έχει το κάθε ξενοδοχείο καθώς και τις φωτογραφίες του ξενοδοχείου

hotels/admin.py

```
class RoomInline(admin.StackedInline):
    model = Room
    extra = 0


class HotelImageInline(admin.TabularInline):
    model = HotelImage
    extra = 0


class HotelAdmin(admin.ModelAdmin):
    #slug is the name of the hotel that appears in the url hyphen(/)
seperated and lowercase
    prepopulated_fields = {"slug": ("name",) }
    inlines = [RoomInline, HotelImageInline, ]
    list_display = ('name', 'address', 'phone', 'category',
'short_description')

    def short_description(self, obj):
        return obj.description[:100]
```

Δημιουργούμε λοιπόν μια νέα κλάση με το όνομα HotelAdmin και μέσα σε αυτή προσθέτουμε τις κλάσεις που επιθυμούμε. Στη συνέχεια για να το δηλώσουμε στο διαχειριστικό πράττουμε όπως και προηγουμένως

```
admin.site.register(Hotel, HotelAdmin)
```

4.3.4 App hotels views

Κάθε view είναι υπεύθυνο στο να κάνει ένα από δύο πράγματα: να επιστρέφει ένα HttpResponse object το οποίο περιέχει το περιεχόμενο για την σελίδα που ζητήθηκε ή να κάνει raise κάποιο exception όπως το Http404. Τα υπόλοιπα εξαρτώνται από εσάς.

Το view σας μπορεί να διαβάσει records από την βάση δεδομένων ή όχι. Μπορεί να χρησιμοποιήσει ένα template system όπως αυτό του Django – ή κάποιο τρίτο Python template system – ή όχι. Μπορεί να παράγει κάποιο PDF αρχείο, να εξάγει XML, να δημιουργεί κάποιο ZIP αρχείο, να εξάγει JSON δεδομένα, βασικά μπορεί να κάνει τα πάντα χρησιμοποιώντας οποιεσδήποτε βιβλιοθήκες Python επιθυμείτε.

Το μόνο που απαιτείται από το Django είναι ένα HttpResponse ή μια exception.

Μέσα στο αρχείο views.py αφού φορτώσουμε τις απαραίτητες βιβλιοθήκες στην δικιά μας περίπτωση οι ακόλουθες

```
from itertools import groupby

from django.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect
from django.urls import reverse
from django.views import generic
from django import template

from .models import Hotel, Room
from .forms import HotelFilterForm
```

Ξεκινάμε να γράφουμε τις function που θα ελέγχουν τις πληροφορίες που θα περνάμε μέσα στο template μας για να βλέπει ο χρήστης. Συνδέουμε τα μοντέλα που έχουν να κάνουν με το view αυτό και κάνουμε τα ερωτήματα στη βάση μας για τις πληροφορίες που θα χρειαστούμε να εμφανίζουμε στο χρήστη.

```
def index(request):

    params = request.GET

    # BUILDING QUERIES
    hotels = Hotel.objects.order_by('name').all()

    data = {
        'hotels': hotels,
        'active_tab': 'hotels',
    }

    return render(request, 'hotels/hotel_list.html', data)
```

Αυτή η function μας επιστρέφει όλα τα ξενοδοχεία που έχουμε στη βάση και τα αποθηκεύει στη μεταβλητή hotels. Στη μεταβλητή params αποθηκεύουμε τις get παραμέτρους αν υπάρχουν Στη συνέχεια δημιουργούμε ένα dictionary το οποίο ονομάζεται data, μέσα στο οποίο περνάμε στη μεταβλητή hotels τα hotels που ζητήσαμε πριν από τη βάση μας. Με αυτόν τον τρόπο μέσω της μεταβλητής hotel μπορούμε να έχουμε πρόσβαση μέσα στα πεδία του μοντέλου πχ hotel.name . Μια ακόμα πληροφορία που περνάμε είναι το active_tab έτσι ρυθμίζουμε πάνω στο navigation ποιο tab θα είναι ενεργό καθώς περιηγούμαστε στις σελίδες.

4.3.5 App hotels URL's

Είναι πολύ σημαντικό να έχουμε καθαρά και ευανάγνωστα url. Το django μας δίνει τη δυνατότητα να δημιουργήσουμε τα url μας όπως μας αρέσει και με καθαρό τρόπο.

Το αρχείο urls.py κρατά όλα τα url για το συγκεκριμένο app

hotels/ulrs.py

```
app_name = 'hotels'
urlpatterns = [
      url(r'^$', views.index, name='index'),
      url(r'^(?P<slug>[\w-]+)/$', views.detail, name='detail'),
]
```

Όπως βλέπουμε από πάνω έχουμε τη δυνατότητα να ονομάζουμε τα url το οποίο είναι πάρα πολύ χρήσιμο σε περίπτωση που θέλουμε να αλλάξουμε το path στο οποίο αναφερόμαστε . Αν δεν υπήρχε αυτό και θέλαμε να κάνουμε κάποια αλλαγή , θα έπρεπε να ανατρέξουμε σε όλα τα σημεία του κώδικα που έχουμε χρησιμοποιήσει αυτό το url και να το αλλάξουμε. Με αυτόν το τρόπο η μόνη αλλαγή που πρέπει να κάνουμε είναι σε ένα σημείο χωρίς να επηρεάζονται τα υπόλοιπα σημεία του κώδικα.

Η πρώτη παράμετρος δέχεται regex τα οποία κάνουν match με τη πληροφορία που θέλουμε να δείχνουμε. Για παράδειγμα το πρώτο είναι κενό , εδώ ερχόμαστε από το kavala/ulrs.py

kavala/ulrs.py

```
urlpatterns = [
      url(r'^$', TemplateView.as_view(template_name='homepage.html'),
name='homepage'),
    url(r'^hotels/',include('hotels.urls')),
    url(r'^rentals/',include('rentals.urls')),
    url(r'^contact/',include('contact.urls')),
    url(r'^sights/',include('sights.urls')),
    url(r'^search/',include('booking.urls')),
    url(r'^settings/',include('user_profile.urls')),
    url(r'^accounts/register/$', MyRegistrationView.as_view(),
name='registration_register'),
    url(r'^accounts/', include('registration.backends.simple.urls')),
    url(r'^admin/', admin.site.urls),
]+ static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Έτσι στο hotels/ ξέρει ότι αντιστοιχεί η function views.index, η οποία με τη σειρά της αφού εκτελέσει τις εντολές που έχουμε ορίσει, επιστρέφει τη σελίδα hotel_list.html με τις πληροφορίες που χρειάζεται.

Το δεύτερο url μας οδηγεί πχ στο hotels/egnatia/ , την πληροφορία αυτή την αντλεί από το regex που κοιτάει το slug πεδίο το οποίο το περνάμε σαν πληροφορία μέσα από το template του hotel_list.html. Στο σημείο που θέλουμε να έχουμε το link για τη σελίδα detail.html έχουμε τον κώδικα

hotel_list.html

```
<a href="{% url 'hotels:detail' hotel.slug %}">{{ hotel.name }}</a>
```

4.3.6 App hotels templates

Ένα template είναι ένα απλό αρχείο κειμένου. Μπορεί να δημιουργηθεί από οποιοδήποτε text-based format (HTML, XML, CSV, κλπ.). Περιέχει μεταβλητές οι οποίες αντικαθιστώνται με τιμές όταν φορτωθεί το template και tags τα οποία ελέγχουν τη λογική του template.

Το django μέσα από το “extends” tag μας δίνει τη δυνατότητα να έχουμε ένα κομμάτι κώδικα ως δομή και να μπορούμε να το αναπαράγουμε σε όσες σελίδες το χρειαζόμαστε αλλάζοντας κάθε φορά δυναμικά το περιεχόμενο που επιθυμούμε. Έτσι έχουμε έναν οργανωμένο κώδικα που δεν επαναλαμβάνεται και η συντήρηση του γίνεται πολύ εύκολη αφού συντηρούμε ένα κομμάτι αντί για πολλά.

Ακόμα έχουμε τη δυνατότητα να δημιουργούμε και δικά μας tag και ανάλογα με τη χρήση που θέλουμε να κάνουμε να το χρησιμοποιούμε . Οι επιλογές είναι αμέτρητες και φυσικά μπορούμε και σε μελλοντικά projects να το επαναχρησιμοποιήσουμε.

Για παράδειγμα το template hotel_list.html μπορεί να μας δώσει μια εικόνα για όσα αναφέρθηκαν

```
{% extends 'layouts/base.html' %}
{% load static %}

{% block title %}Kavala'sHotels{% endblock %}

{% block header %}
{% include 'partials/pages-header.html'  with page_title="kavala's
hotels" page_subtitle="find the best choice for you" %}
{% endblock %}
```

Ακόμα έχουμε την δυνατότητα να κάνουμε επαναλήψεις για να πάρουμε όλες τις πληροφορίες από μια μεταβλητή καθώς και ελέγχους ακεραιότητας

```
{% for hotel in hotels %}
      {% if hotel.image %}
            -Κάνε κάτι-
      {% else %}
            -Κάνε κάτι άλλο-
      {% endif %}
{% endfor %}
```

Για να πάρουμε δυναμικά ονόματα μέσα από μεταβλητές χρησιμοποιούμε το {{}}

```
πχ {{ hotel.name  }}
```

4.3.7 App Contact και Forms

Στην HTML, μια φόρμα είναι μια συλλογή των στοιχείων στο εσωτερικό <form> ... </ form> που επιτρέπουν σε ένα επισκέπτη να κάνει πράγματα όπως την εισαγωγή κειμένου, να επιλέξει από τις επιλογές, να χειριστεί αντικείμενα ή ελέγχους, και ούτω καθεξής, και στη συνέχεια να στείλει αυτές τις πληροφορίες πίσω στο διακομιστή.

Μερικά από αυτά τα στοιχεία διεπαφής elements - text input της φόρμας είναι αρκετά απλά και είναι ενσωματωμένα στο ίδιο το HTML. Άλλα είναι πολύ πιο περίπλοκα, μια διεπαφή που αναδύεται όπως ένα εργαλείο επιλογής ημερομηνίας ή σας επιτρέπει να μετακινήσετε ένα ρυθμιστικό, χρησιμοποιούν συνήθως JavaScript και CSS, καθώς και HTML φόρμα <input> στοιχεία για την επίτευξη αυτών των αποτελεσμάτων

Καθώς είναι <input> τα στοιχεία, μια φόρμα πρέπει να καθορίζει δύο πράγματα:

που: το URL στο οποίο τα δεδομένα που αντιστοιχούν στην είσοδο του χρήστη θα πρέπει να επιστραφούν
πώς: η μέθοδος HTTP που τα δεδομένα θα πρέπει να επιστρέφονται

Η λειτουργικότητα φορμών του Django μπορεί να απλοποιήσει και να αυτοματοποιήσει τεράστια τμήματα του έργου αυτού. Μπορεί επίσης να το κάνει με μεγαλύτερη ασφάλεια από ό, τι οι περισσότεροι προγραμματιστές θα είναι σε θέση να κάνουν σε κώδικα που οι ίδιοι έγραψαν.

Το Django χειρίζεται τρία διακριτά μέρη του έργου που εμπλέκονται με τις φόρμες:

- την προετοιμασία και την αναδιάρθρωση των δεδομένων ώστε να είναι έτοιμο για το rendering
- τη δημιουργία φορμών HTML για τα δεδομένα
- τη λήψη και την επεξεργασία των submitted φορμών του χρήστη

Στο app contact είναι ένα χαρακτηριστικό παράδειγμα από τη χρήση του django forms που έγινε σε αυτή τη σελίδα, όπου τα πεδία της φόρμας δημιουργούνται μέσα στο αρχείο forms.py και έρχονται δυναμικά στη σελίδα μέσα από το views.py. Σε εκείνο το σημείο γίνεται και η επεξεργασία των πληροφοριών καθώς και η αποθήκευσή τους.

contact/forms.py

```
class ContactForm(forms.Form):
    error_css_class = 'error'
    required_css_class = 'required'

    fname = forms.CharField(label='First Name', max_length=100,
       widget=forms.TextInput(attrs={'placeholder': 'Enter Name'}))
    lname = forms.CharField(label='Last Name', max_length=100,
       widget=forms.TextInput(attrs={'placeholder': 'Enter Last Name'}))
    email = forms.EmailField(label='Email',
       widget=forms.TextInput(attrs={'placeholder': 'Enter Email'}))
    phone = forms.CharField(label='Phone', max_length=20,
       widget=forms.TextInput(attrs={'placeholder': 'Enter Phone'}), required=False)
    message = forms.CharField(label='Message',
       widget=forms.Textarea(attrs={'placeholder': 'Write your message'}))

    def save(self):
        return Contact.objects.create(fname=self.cleaned_data['fname'],
       lname=self.cleaned_data['lname'], email=self.cleaned_data['email'],
       phone=self.cleaned_data['phone'], message=self.cleaned_data['message'])
```

4.4 Σύστημα ελέγχου ταυτότητας χρήστη (User authentication)

Το Django έρχεται με ένα σύστημα ελέγχου ταυτότητας χρήστη. Χειρίζεται τους λογαριασμούς χρηστών, ομάδων, τα δικαιώματα και τις συνεδρίες των χρηστών που βασίζονται σε cookies.

Το σύστημα ελέγχου ταυτότητας Django χειρίζεται τόσο την πιστοποίηση όσο και την εξουσιοδότηση . Εν συντομία, ο έλεγχος ταυτότητας επαληθεύει αν ένας χρήστης είναι αυτός που ισχυρίζεται ότι είναι και η εξουσιοδότηση καθορίζει τις δυνατότητες που έχει. Εδώ ο όρος authentication χρησιμοποιείται για να αναφέρεται και στις δύο εργασίες.

Το σύστημα auth αποτελείται από:

- Χρήστες
- Δικαιώματα: Binary σημαίες (ναι / όχι) που ορίζει το αν ένας χρήστης μπορεί να εκτελέσει μια συγκεκριμένη εργασία.
- Ομάδες: Ένας γενικός τρόπος χορήγησης σημάτων και δικαιωμάτων σε περισσότερους από έναν χρήστες.
- Ένα διαμορφώσιμο σύστημα κατακερματισμού κωδικού πρόσβασης
- Forms και view εργαλεία για τη σύνδεση σε χρήστες, ή τον περιορισμό περιεχομένου
- Ένα συνδεόμενο σύστημα backend

Το σύστημα ελέγχου ταυτότητας στο Django έχει ως στόχο να είναι πολύ γενικό και δεν παρέχει κάποια χαρακτηριστικά που βρίσκονται συνήθως σε συστήματα ελέγχου ταυτότητας web. Οι λύσεις για ορισμένα από αυτά τα κοινά προβλήματα έχουν εφαρμοστεί σε πακέτα τρίτου κατασκευαστή:

- Έλεγχο αντοχής κώδικα
- Στραγγαλισμός των προσπαθειών σύνδεσης
- Authentication κατά τρίτους (OAuth, για παράδειγμα)

Μέσα στο app user_profile έχουμε δημιουργήσει το μοντέλο μας βασισμένο στο Authentication του Django. Χρειάζετε να κάνουμε χρήση της βιβλιοθήκης django.contrib.auth.models το μοντέλο User

user_profile/models.py

```
# Create your models here.
class UserProfile(models.Model):
    # This line is required. Links UserProfile to a User model instance.
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    # The additional attributes we wish to include.
    home_address = models.CharField(max_length=50, blank=True)
    phone_numer = models.CharField(max_length=50, blank=True)
    picture = models.ImageField(upload_to='profile_images', blank=True)

    # Override the __unicode__() method to return out something
meaningful!
    def __str__(self):
        return self.user.username
```

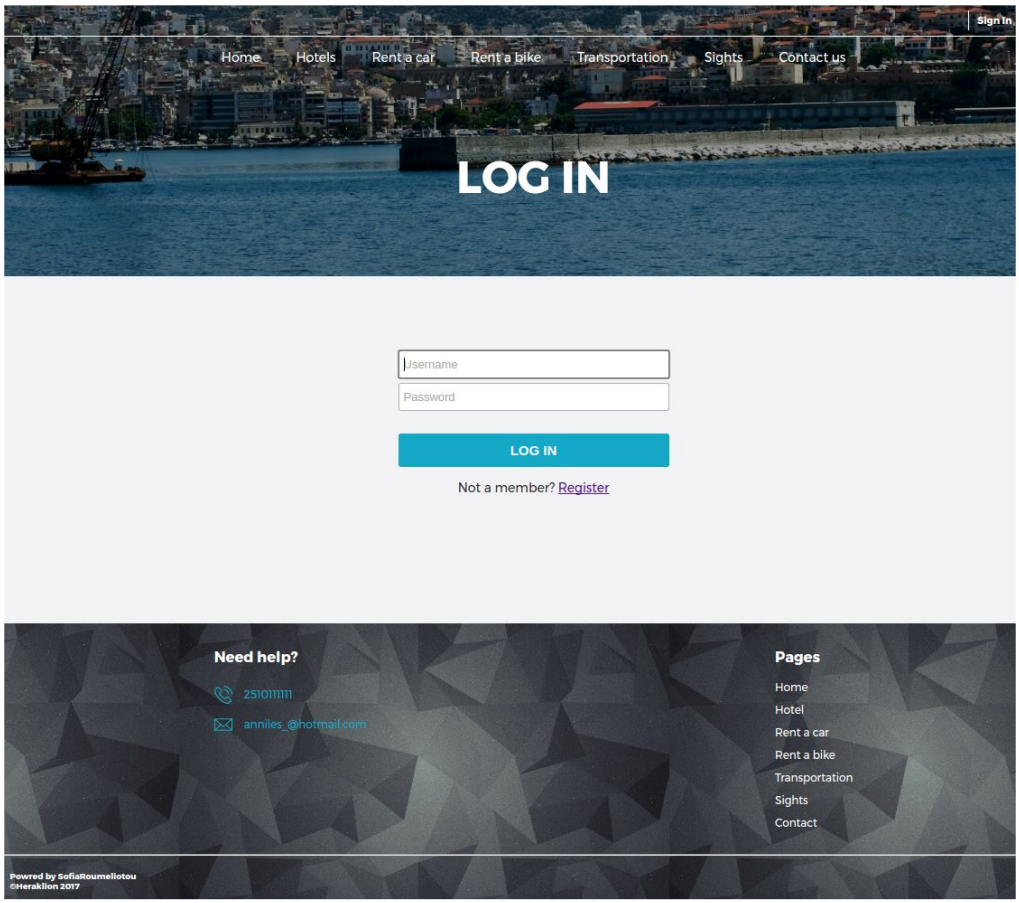
Αν θέλουμε να ελέγχουμε την προσβασιμότητα που θα έχει ένας χρήστης ανάλογα με το αν έχει συνδεθεί ή όχι, μπορούμε να το πετύχουμε πολύ εύκολα με τη χρήση των decorators (@login_required) ακριβώς πρίν από τις functions που θέλουμε να το εφαρμόσουμε.

4.4.1 Login

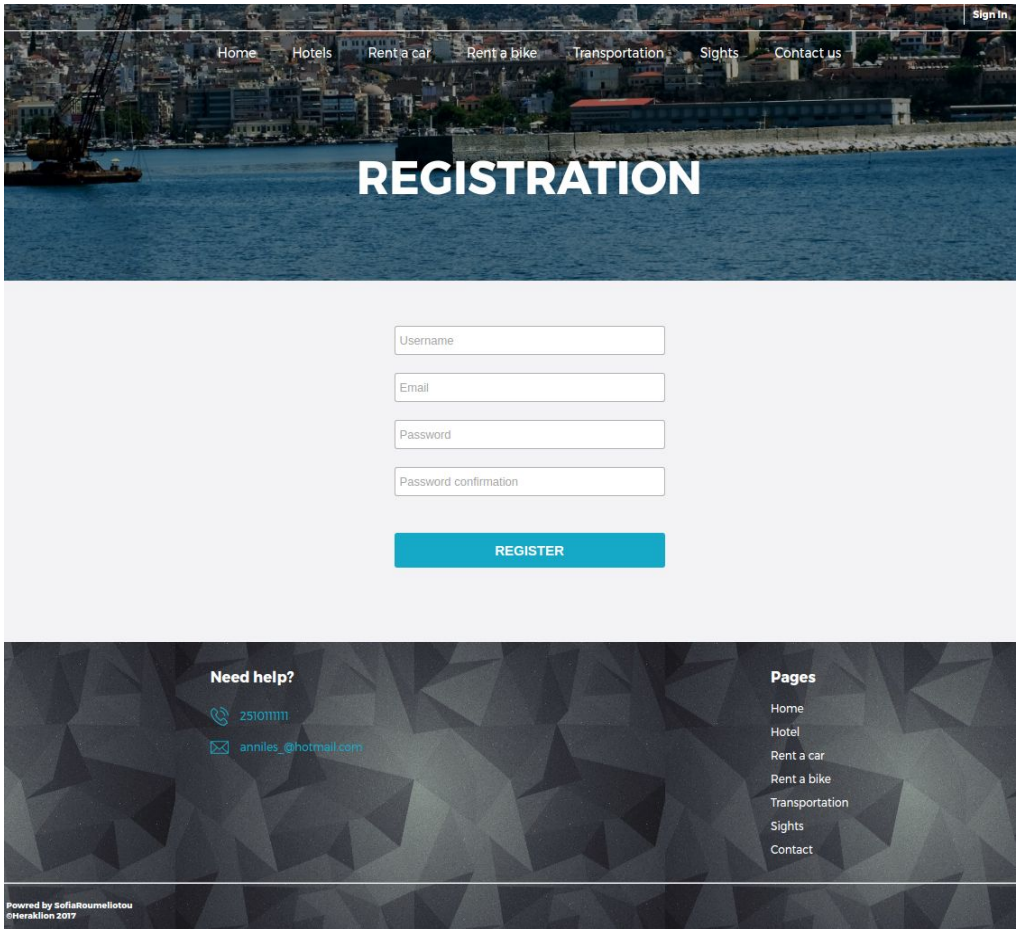
Για την διαχείριση των χρηστών όσο αφορά το login, logout και registration έγιναν με την εφαρμογή django-registration-redux . Η προεπιλεγμένη ρύθμιση θα επιτρέψει την εγγραφή του χρήστη με την ακόλουθη ροή εργασίας

- Ένας χρήστης κάνει signs up για έναν λογαριασμό με την παροχή ενός ονόματος χρήστη, διεύθυνση ηλεκτρονικού ταχυδρομείου και τον κωδικό πρόσβασης.
- Από αυτές τις πληροφορίες, ένα νέο αντικείμενο χρήστη έχει δημιουργηθεί, με το is_active πεδίο να έχει οριστεί σε False. Επιπλέον, ένα κλειδί ενεργοποίησης παράγεται και αποθηκεύεται, και ένα μήνυμα ηλεκτρονικού ταχυδρομείου αποστέλλεται στο χρήστη που περιέχει ένα σύνδεσμο για να κάνει κλικ και να ενεργοποιήσει το λογαριασμό του.
- Μόλις κάνει κλικ στο σύνδεσμο ενεργοποίησης, ο νέος λογαριασμός γίνεται ενεργός (το is_active πεδίο έχει οριστεί σε True) Μετά από αυτό, ο χρήστης μπορεί να συνδεθεί.

Στιγμιότυπο από τη σελίδα login για τους χρήστες



Στιγμιότυπο από τη σελίδα register

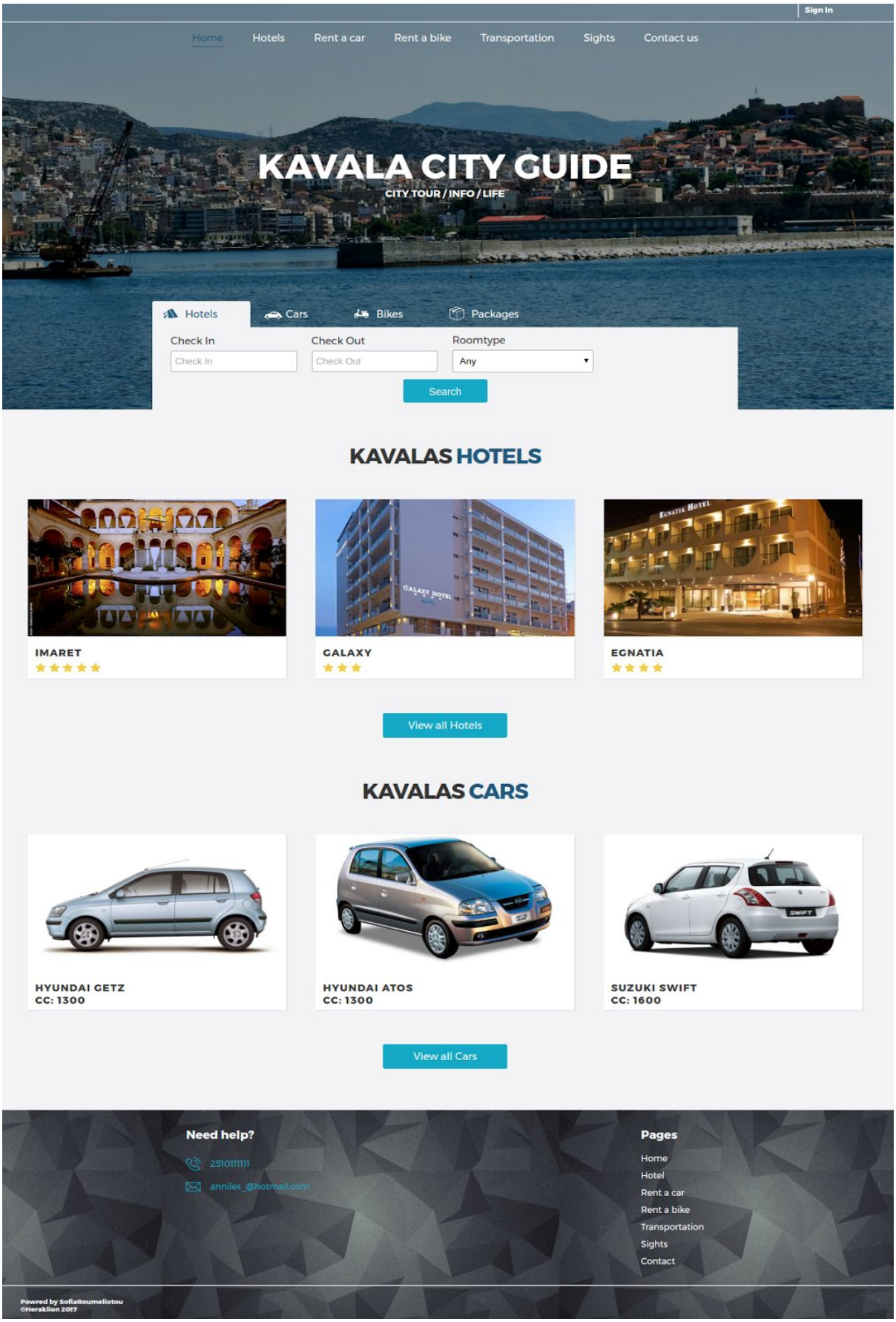


4.5 Περιγραφή Εφαρμογής

Μπαίνοντας ο χρήστης στη σελίδα αρχικά βέπει το navigation menu με το οποίο μπορεί να περιηγηθεί στη σελίδα. Μέσα στο header υπάρχει μια φόρμα αναζήτησης ανάλογα με το είδος που ενδιαφέρεται ο χρήστης το οποίο μπορεί να είναι ξενοδοχείο, αυτοκίνητο ή μηχανή καθώς και πακέτα . Μπορεί να επιλέξει τις ημερομηνίες που τον ενδιαφέρουν και στη συνέχει να μεταφερθεί στη λίστα όπου θα φαίνονται οι αντίστοιχες επιλογές.

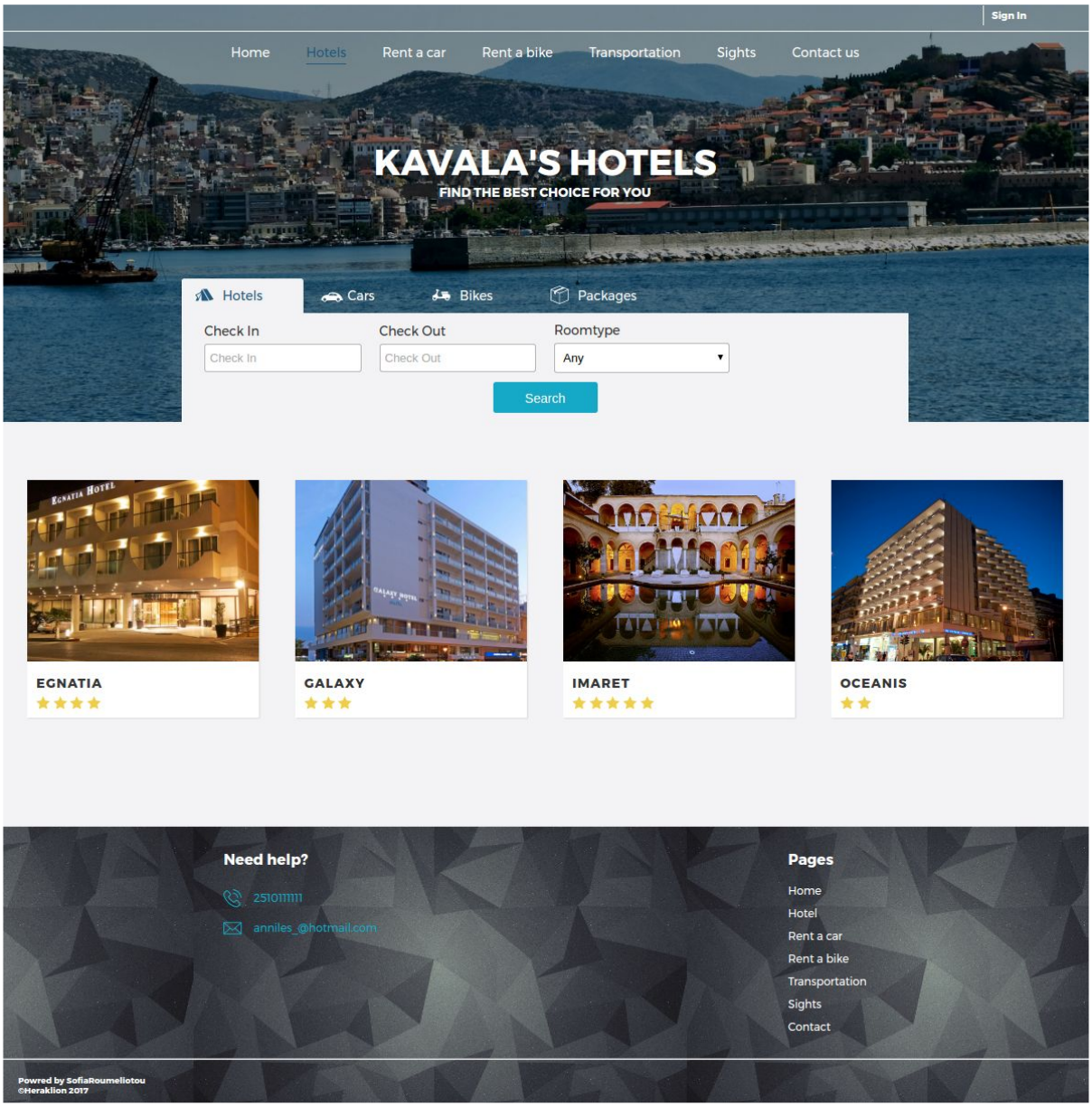
Στη σελίδα με την λίστα της αναζήτησης υπάρχουν και άλλες επιλογές για να μπορέσει να κάνει πιο συγκεκριμένη αναζήτηση σύμφωνα με τις επιθυμίες του. Από εκείνο το σημείο αφού ο χρήστης έχει καταλήξει μπορεί να μπει στη σελίδα του κάθε αντικειμένου για να δει εικόνες και περισσότερες πληροφορίες.

Ακολουθούν εικόνες με την παρουσίαση των σελίδων που αναφέραμε.


Sign In
Home
Hotels
Rent a car
Rent a bike
Transportation
Sights
Contact us
KAVALA CITY GUIDE
CITY TOUR / INFO / LIFE
Hotels
Cars
Bikes
Packages
Check In
Check Out
Roomtype
Any
Search
KAVALAS HOTELS
IMARET
GALAXY
EGNATIA
View all Hotels
KAVALAS CARS
HYUNDAI GETZ
CC: 1300
HYUNDAI ATOS
CC: 1300
SUZUKI SWIFT
CC: 1600
View all Cars
Need help?
Pages
Home
Hotel
Rent a car
Rent a bike
Transportation
Sights
Contact
Powred by SofiaRoumeliotou
©Heraklion 2017

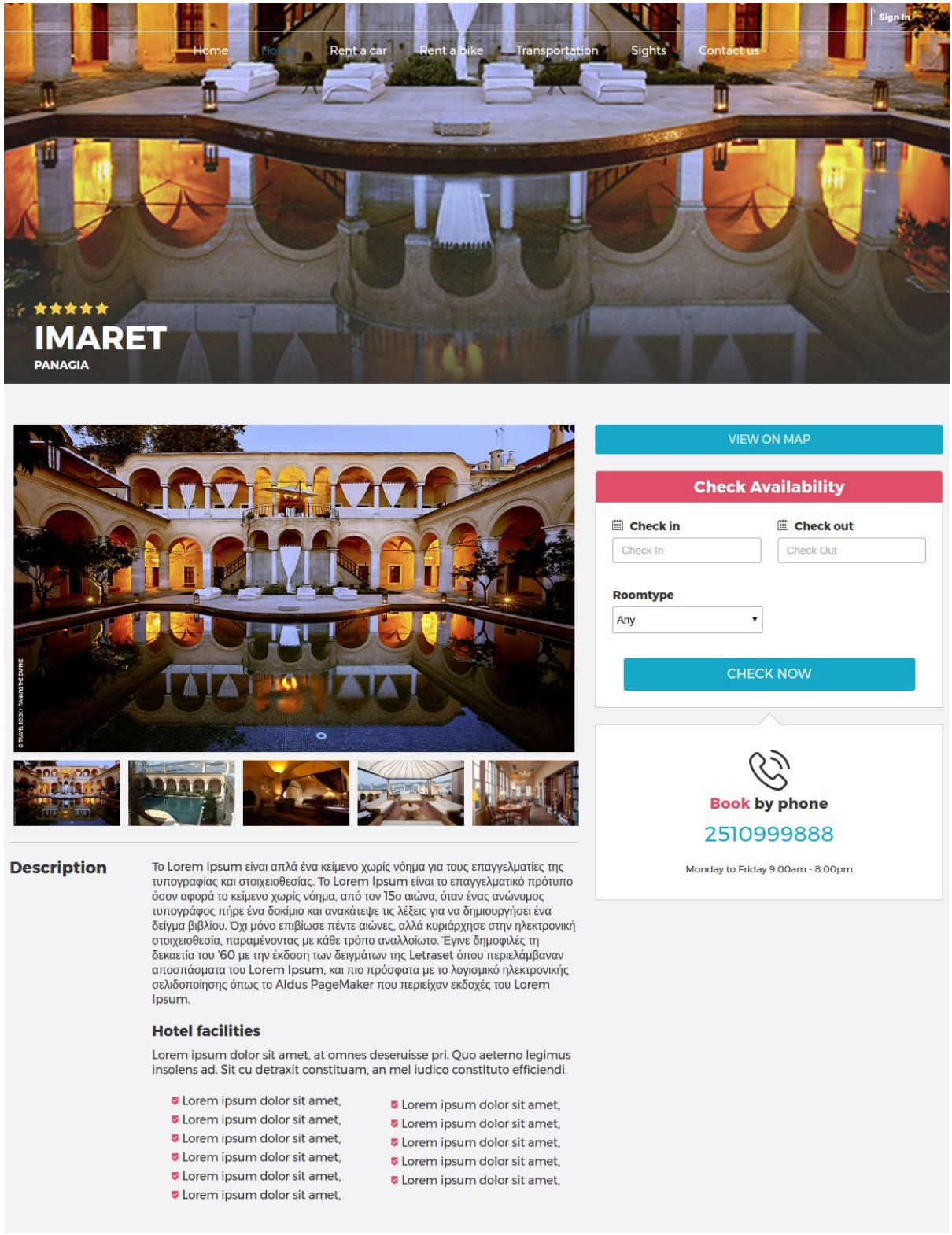
2.Home page

Σε περίπτωση που δεν κάνουμε αναζήτηση και θέλουμε να δούμε στα όλα τα ξενοδοχεία χωρίς πολλές πληροφορίες μπαίνουμε μέσα από το navigation menu στην ακόλουθη σελίδα.

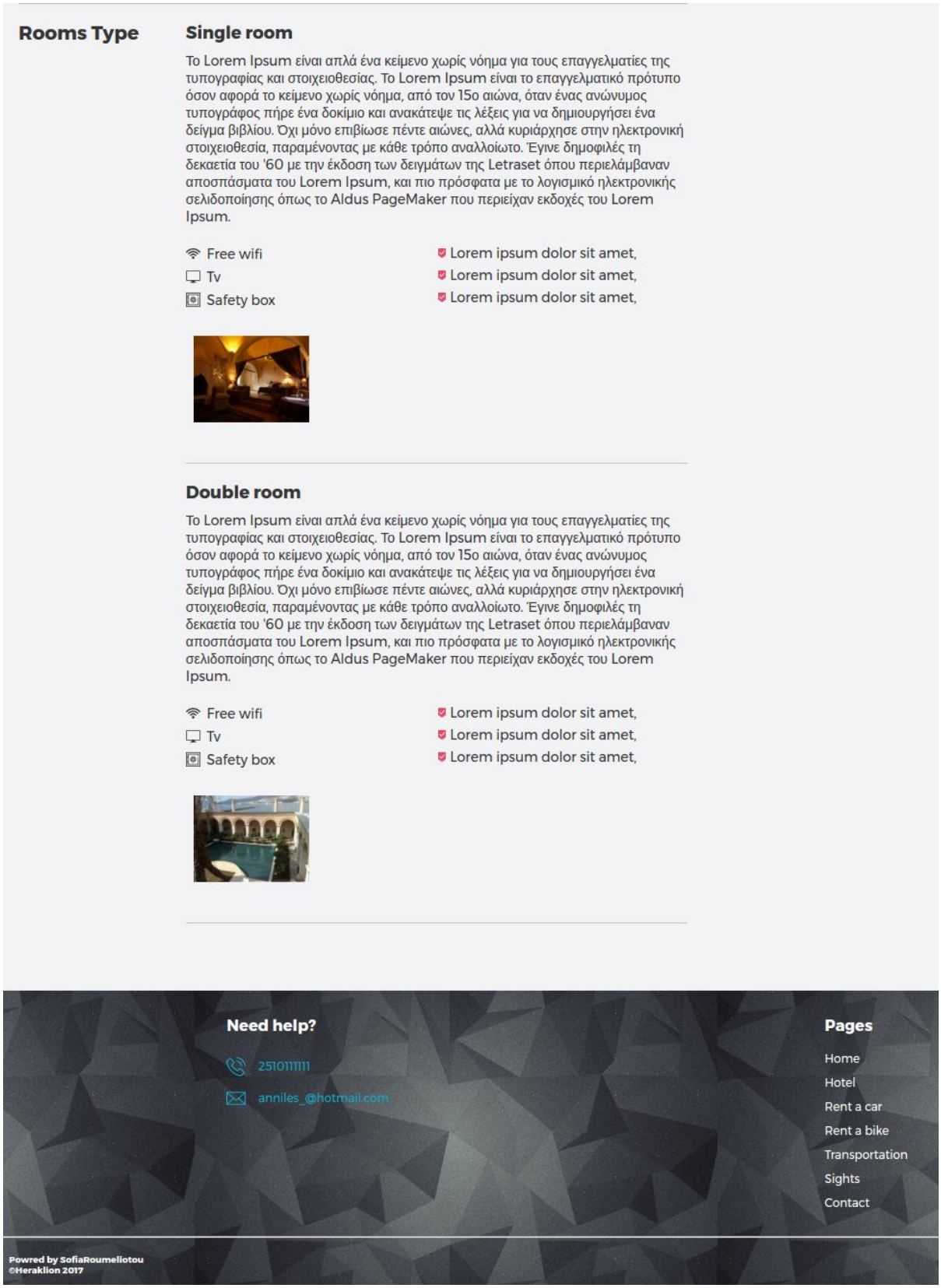


Από αυτό το σημείο αν θέλουμε μπορούμε να κάνουμε την αναζήτηση μας για τις μέρες που μας ενδιαφέρουν. Αλλιώς μπορούμε να μπούμε και μέσα στη σελίδα κάποιου ξενοδοχείου που μας ενδιαφέρει για να πάρουμε μια καλύτερη εικόνα. Παρακάτω είναι ένα παράδειγμα της σελίδας.

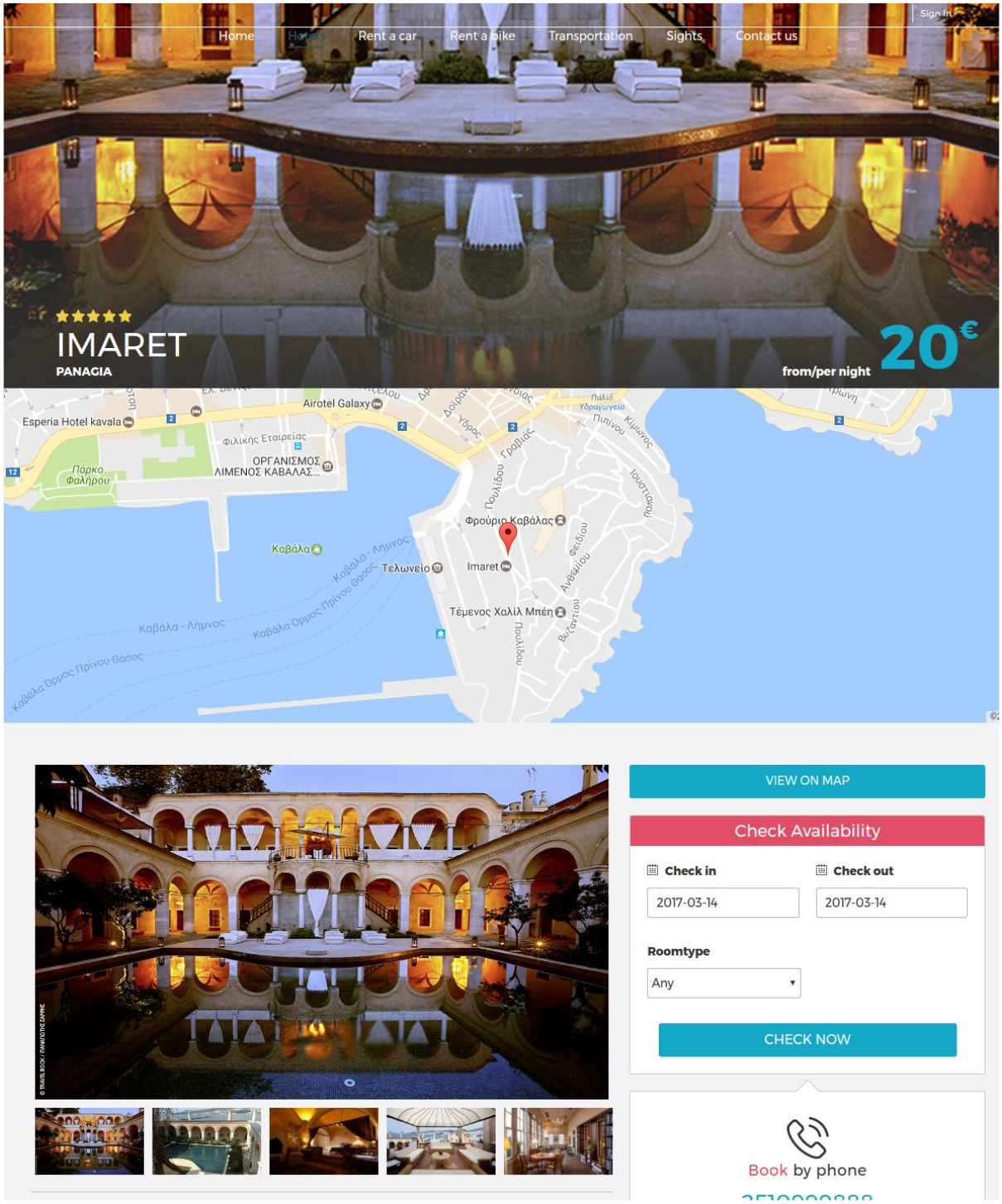
Εδώ βλέπουμε την περιγραφή του ξενοδοχείου τις παροχές που προσφέρει καθώς και ένα slider με εικόνες από τους χώρους του ξενοδοχείου, ώστε να μπορέσουμε να έχουμε μία καλύτερη εικόνα του χώρου και των παροχών.



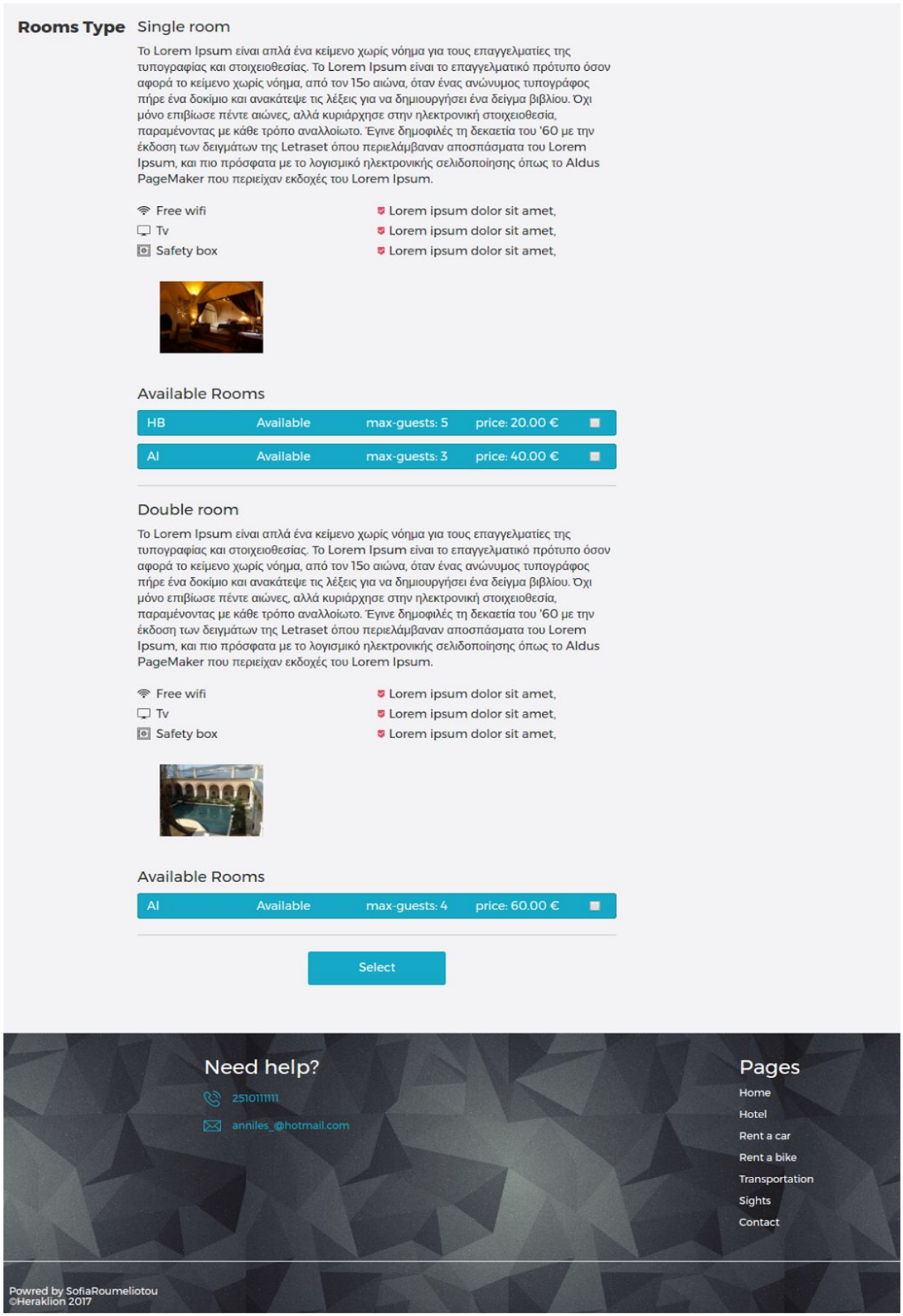
Ακριβώς από κάτω εμφανίζονται τα δωμάτια που διαθέτει το συγκεκριμένο ξενοδοχείο χωρίς να βλέπουμε ακόμα τη διαθεσιμότητα σε αυτά. Έχει μικρές φωτογραφίες των δωματίων καθώς και μια μικρή περίληψη για αυτά.



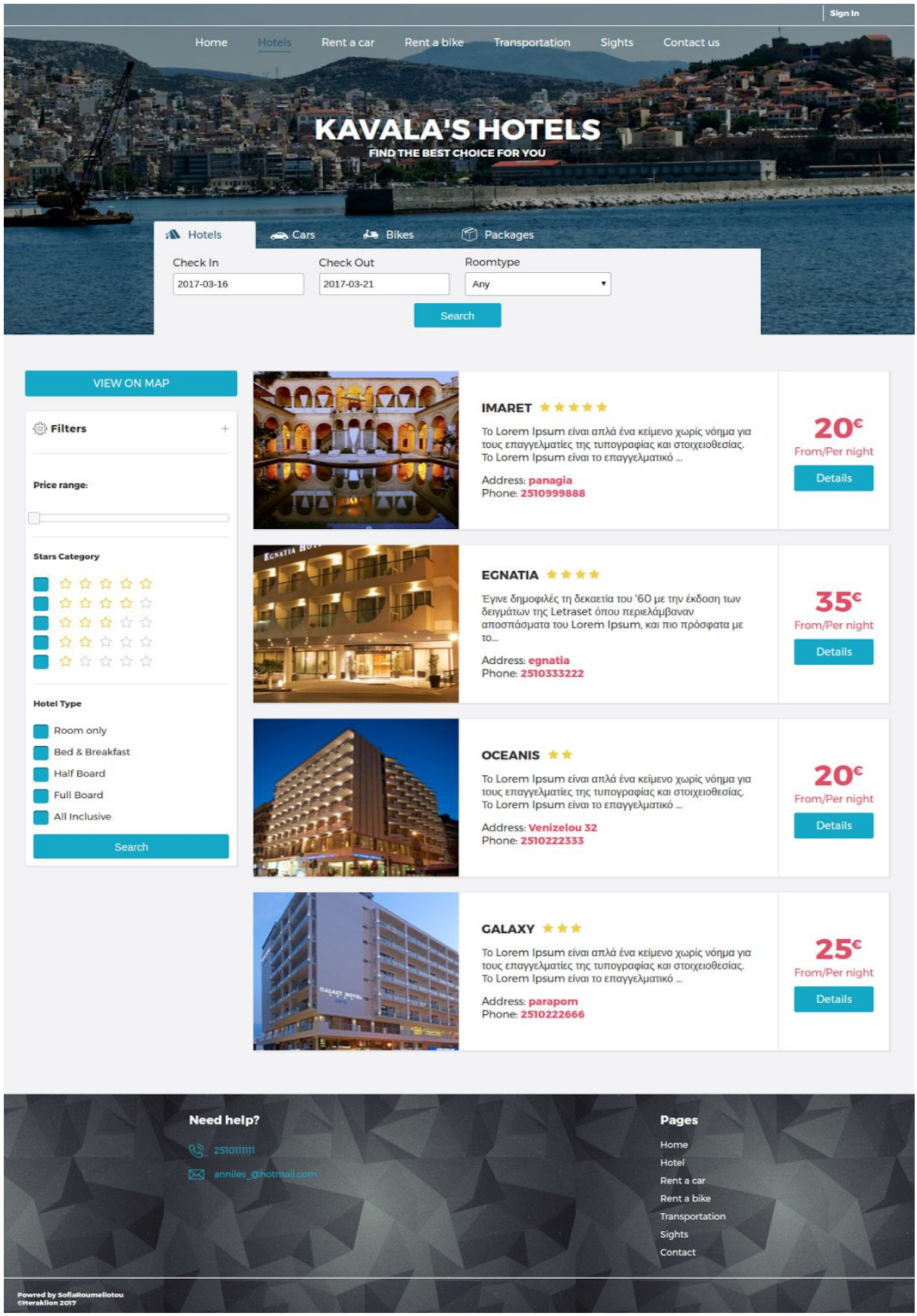
Μέχρι αυτό το σημείο δεν έχει γίνει αναζήτηση διαθεσιμότητας των δωματίων. Μπορούμε όμως να κάνουμε αναζήτηση από τη φόρμα που υπάρχει στα δεξιά της σελίδας και να μας επιστρέψει τα διαθέσιμα δωμάτια. Ακόμα μπορούμε να δούμε μέσα από τους χάρτες του google maps σε ποιο σημείο της πόλης βρίσκετε.



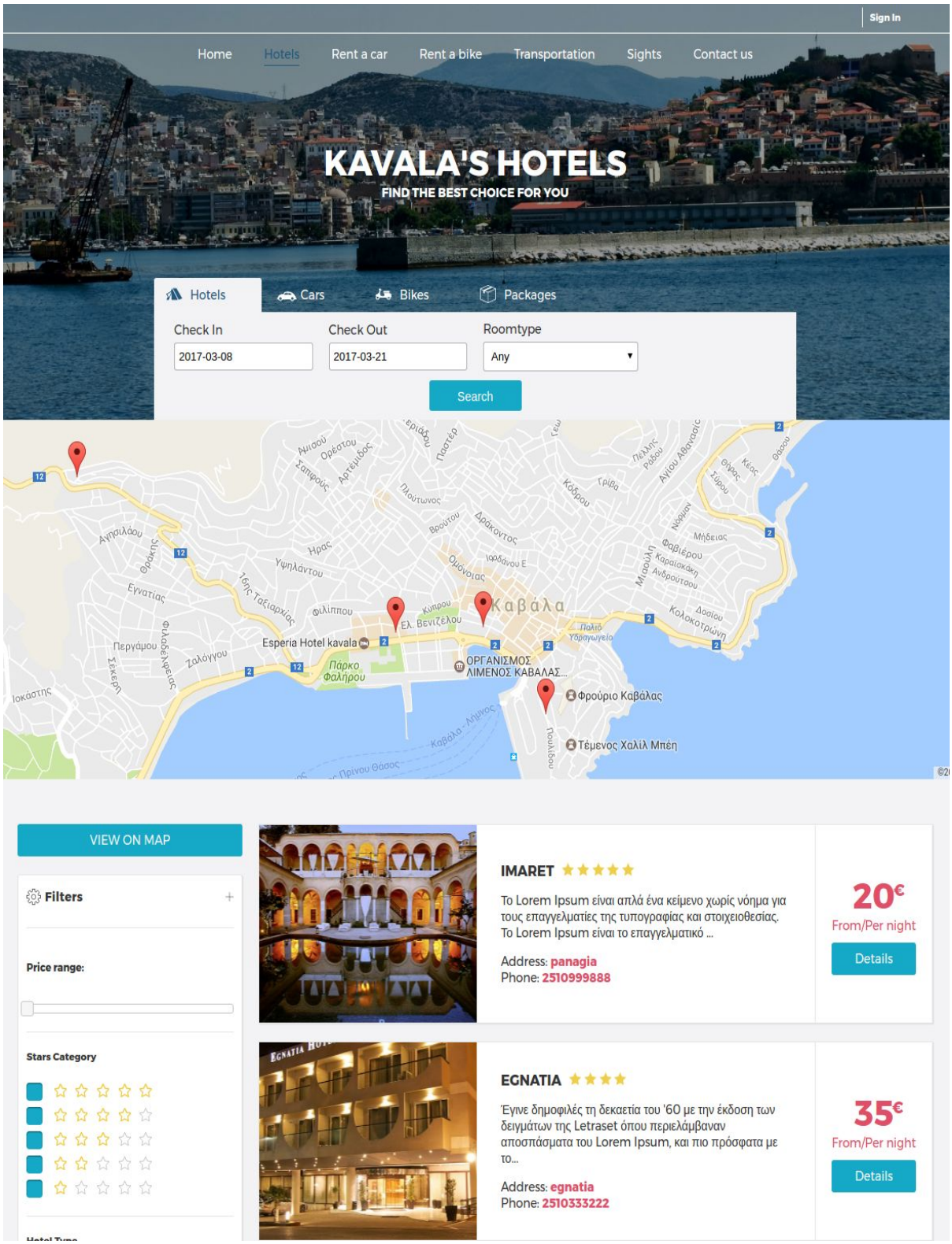
Κάνοντας αναζήτηση στο συγκεκριμένο ξενοδοχείο πήραμε τα συγκεκριμένα αποτελέσματα για τη διαθεσιμότητα των δωματίων του.



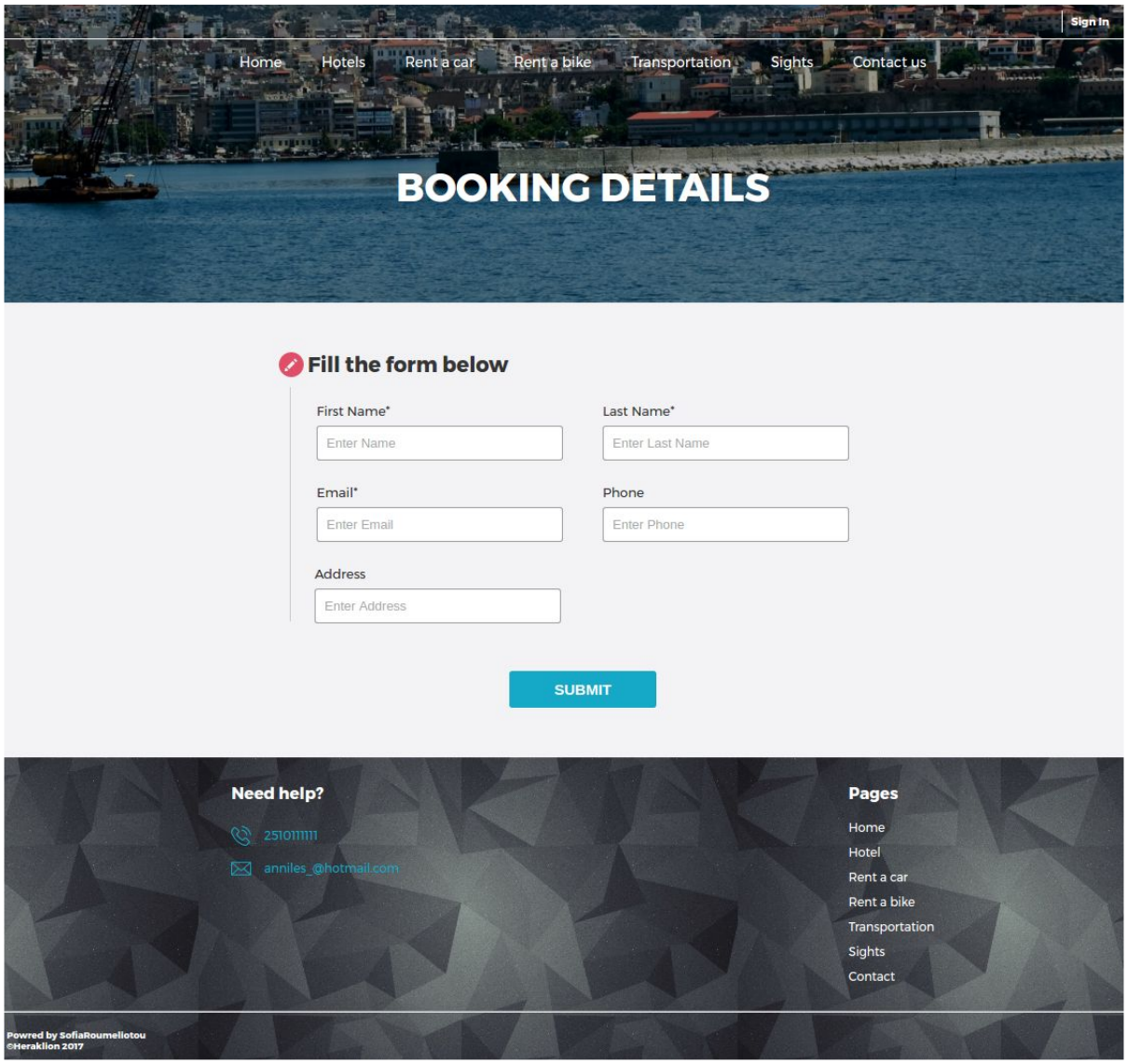
Αν κάναμε αναζήτηση ημερομηνιών από τη φόρμα αναζήτησης που έχουμε στο header η σελίδα με που θα μας επέστρεφε την λίστα των επιλογών που είναι διαθέσιμα.



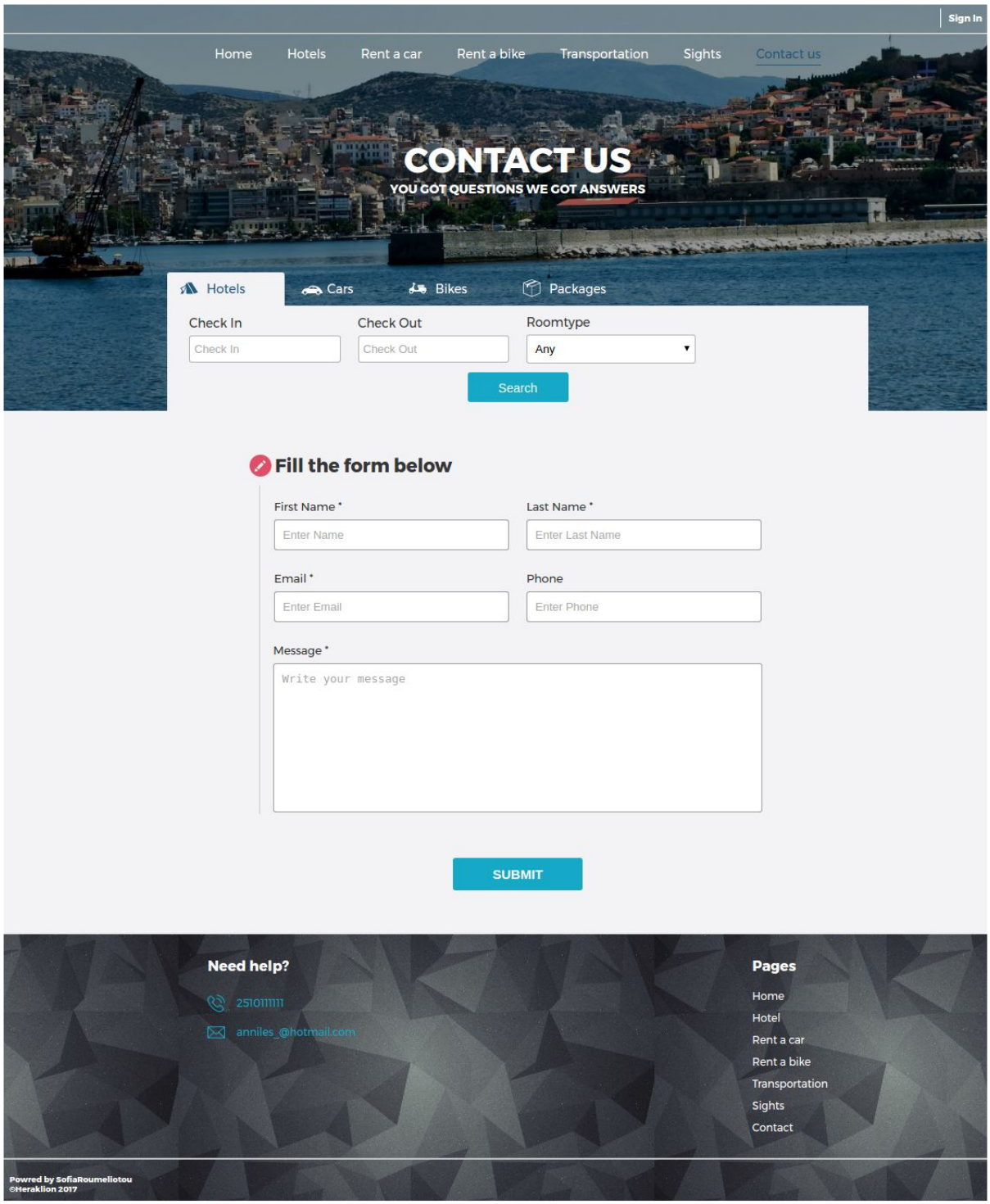
Και σε αυτό το σημείο μπορούμε να δούμε στον χάρτη τις τοποθεσίες των ξενοδοχείων για να έχουμε εικόνα της επιλογής που θα κάνουμε.



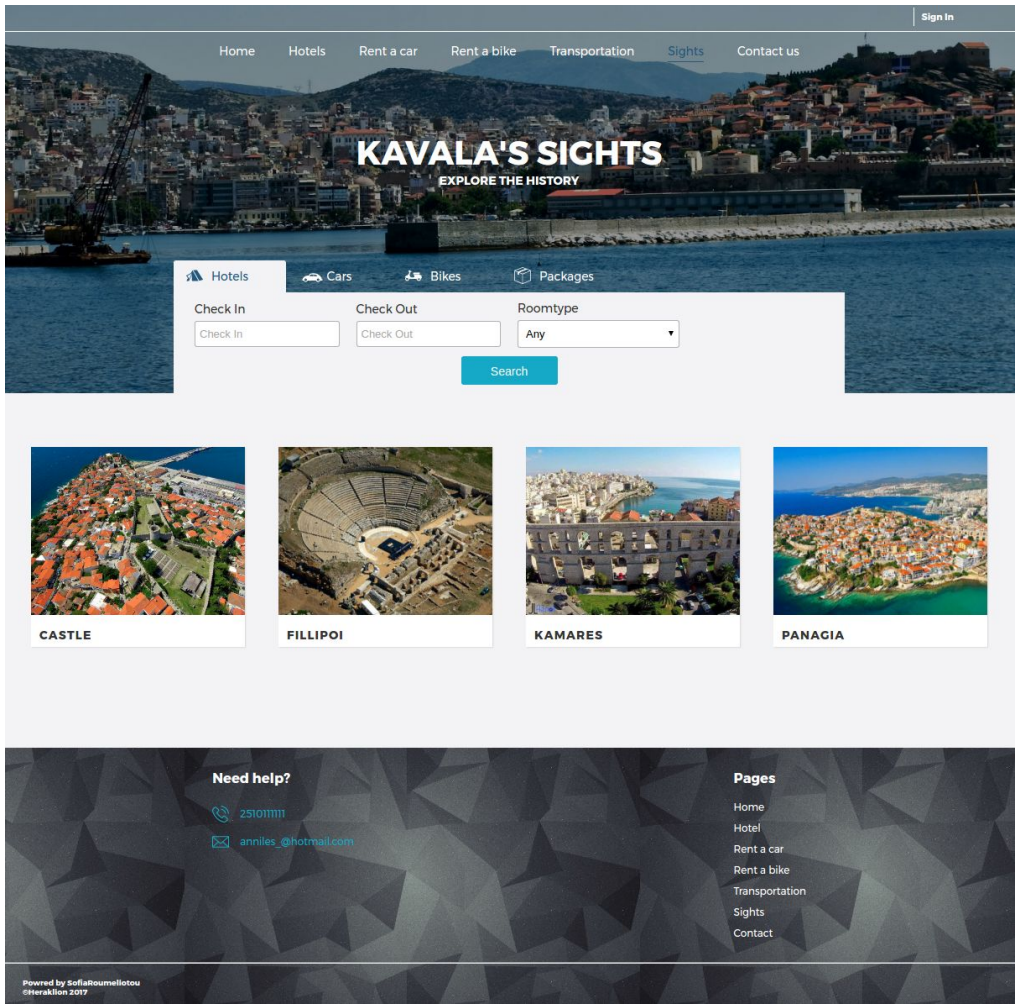
Η ίδια λογική υπάρχει και για τις σελίδες των αυτοκινήτων καθώς και των μηχανών. Αφού έχουμε επιλέξει και θέλουμε να κάνουμε την κράτηση πατώντας να επιλέξουμε μας πάει στη σελίδα για να ολοκληρώσουμε την κράτηση.

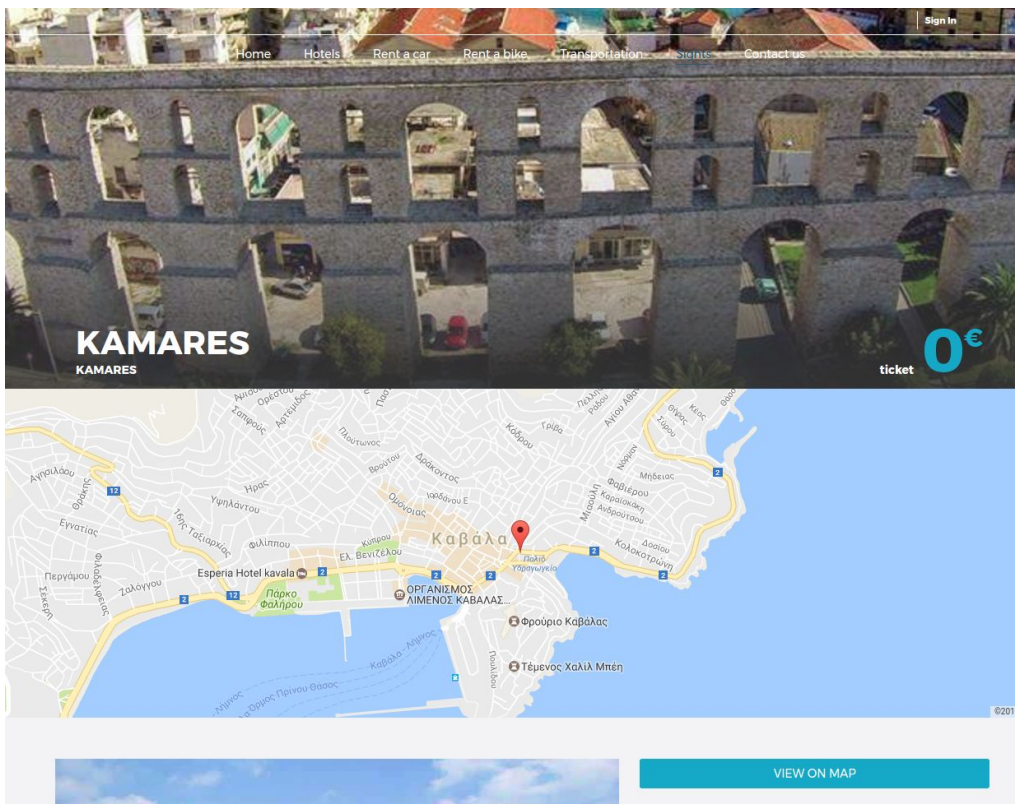


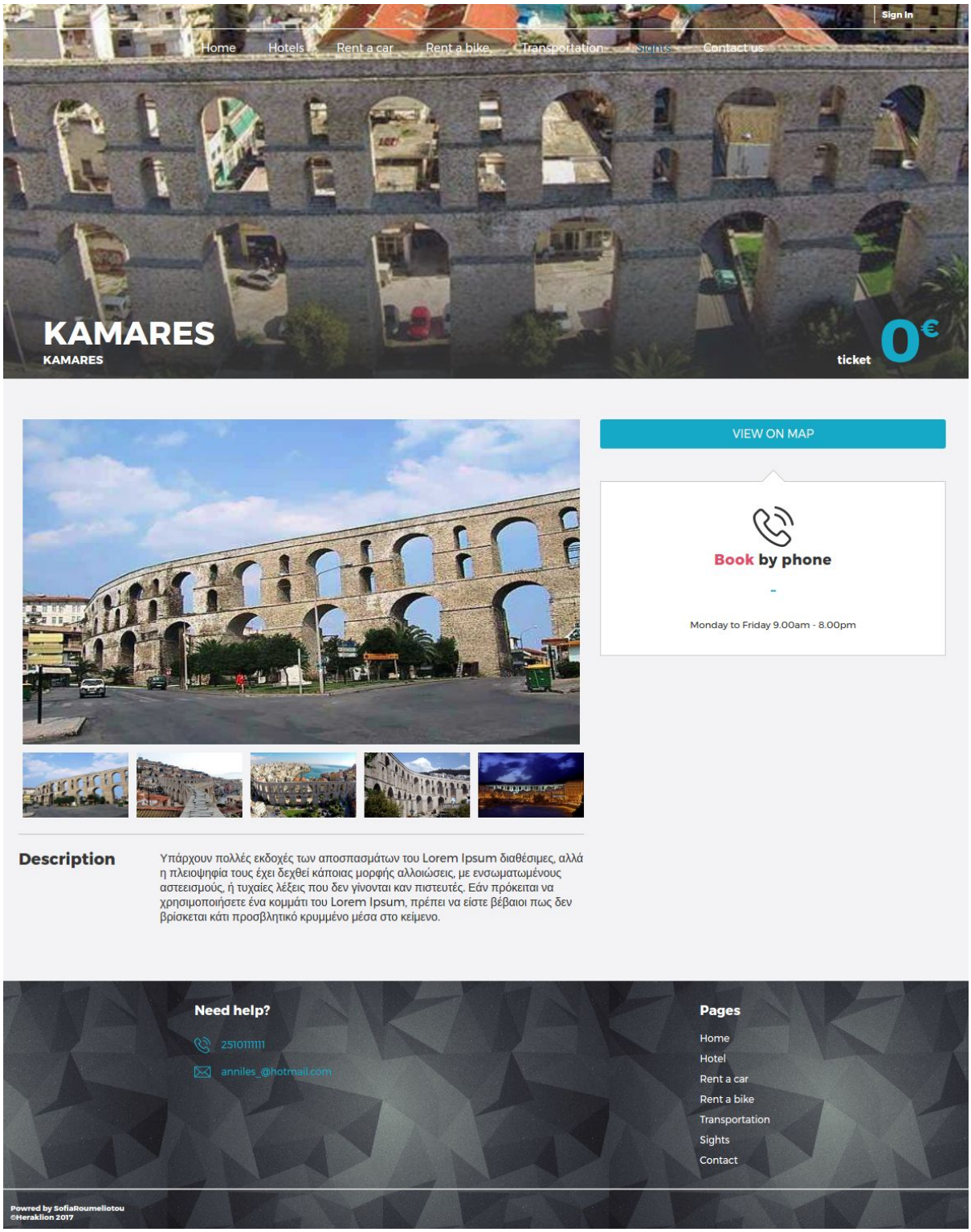
Συμπληρώνοντας τα στοιχεία της φόρμας μπορούμε να ολοκληρώσουμε την κράτηση και να δούμε τις πληροφορίες σχετικά με αυτή. Ακόμα αν θέλουμε να επικοινωνήσουμε με τους διαχειριστές υπάρχει η δυνατότητα μέσα από την φόρμα επικοινωνίας πάνω στο navigation menu.



Για να δούμε τις επιλογές των αξιοθέατων της πόλης μπορούμε να μπούμε μέσα από το μενού όπου θα μας εμφανίσει μια λίστα τα με όσα υπάρχουν. Μπορούμε να δούμε περισσότερες πληροφορίες για το κάθε αξιοθέατο μπαίνοντας μέσα όπου υπάρχει φωτογραφικό υλικό και λίγα λόγια για το κάθε μέρος. Ακολουθούν εικόνες για την αναπαράσταση.


Sign In
Home
Hotels
Rent a car
Rent a bike
Transportation
Sights
Contact us
KAVALA'S SIGHTS
EXPLORE THE HISTORY
Hotels
Cars
Bikes
Packages
Check In
Check Out
Roomtype
Any
Search
CASTLE
FILLIPOI
KAMARES
PANAGIA
Need help?
2510111111
anniles_@hotmail.com
Pages
Home
Hotel
Rent a car
Rent a bike
Transportation
Sights
Contact
Powred by SofiaRoumellotou
©Heraklion 2017


Sign In
Home
Hotels
Rent a car
Rent a bike
Transportation
Sights
Contact us
KAMARES
KAMARES
ticket
0€
Καβάλα
Esperia Hotel kavala
ΟΡΓΑΝΙΣΜΟΣ ΛΙΜΕΝΟΣ ΚΑΒΑΛΑΣ...
Φρούριο Καβάλας
Τέμενος Χαλίλ Μπέη
VIEW ON MAP


Sign In
Home
Hotels
Rent a car
Rent a bike
Transportation
Sights
Contact us
KAMARES
KAMARES
ticket
0€
VIEW ON MAP
Book by phone
-
Monday to Friday 9.00am - 8.00pm
Description
Υπάρχουν πολλές εκδοχές των αποσπασμάτων του Lorem Ipsum διαθέσιμες, αλλά η πλειοψηφία τους έχει δεχθεί κάποιας μορφής αλλοιώσεις, με ενσωματωμένους αστεισμούς, ή τυχαίες λέξεις που δεν γίνονται καν πιστευτές. Εάν πρόκειται να χρησιμοποιήσετε ένα κομμάτι του Lorem Ipsum, πρέπει να είστε βέβαιοι πως δεν βρίσκεται κάτι προσβλητικό κρυμμένο μέσα στο κείμενο.
Need help?
2510111111
anniles_@hotmail.com
Pages
Home
Hotel
Rent a car
Rent a bike
Transportation
Sights
Contact
Powred by SofiaRoumeliotou
©Heraklion 2017

5. Αποτελέσματα

5.1. Συμπεράσματα

Η σελίδα αυτή έχει σχεδιαστεί με μελέτη και προσοχή και προσφέρει ένα ολοκληρωμένο αποτέλεσμα σχετικά με πληροφορίες για την πόλη της Καβάλας. Μπορεί να καλύψει τις βασικές ανάγκες για ένα χρήστη προσφέροντάς του επιλογές και ενημέρωση για της ανάγκες που μπορεί να υπάρξουν. Ακόμα και κάτοικοι της πόλης μπορούν να ενημερωθούν σχετικά με τα ξενοδοχεία, τα καταστήματα ενοικιάσεων, αξιοθέατα και άλλα.

5.2. Μελλοντική Εργασία και Επεκτάσεις

Μια μελλοντική επέκταση του οδηγού αυτού θα μπορούσε να είναι, η συνεργασία με εταιρείες οι οποίες ενδιαφέρονται να διαφημιστούν μέσα από τον οδηγό αυτό και να τον διαφημίσουν και οι ίδιοι ώστε να μπορέσουμε να κεντρίσουμε μεγαλύτερο τουριστικό ενδιαφέρον ως προς την πόλη αυτή. Ακόμα μια ιδέα θα ήταν να ενσωματωθεί μια εφαρμογή μέσα από την οποία θα μπορούσε ο χρήστης να πραγματοποιεί κρατήσεις και για ταξί. Ένα ακόμα σημαντικό κομμάτι, το οποίο θα το έκανε ακόμα πιο χρήσιμο και για τους μόνιμους κατοίκους της Καβάλας, θα ήταν πληροφορίες σχετικά με τα εφημερεύοντα φαρμακεία, νοσοκομεία ακόμα και πρατήρια της περιοχής. Ίσως και η συνεργασία με τις γειτονικές πόλεις της περιοχής θα μπορούσε να συμβάλει ουσιαστικά στο να προβληθεί ο τόπος περισσότερο και να επεκταθεί η εφαρμογή μας συναργατικά.

Βιβλιογραφία

1. http://www.studytonight.com/dbms/database-normalization.php
2. https://git-scm.com/
3. http://sass-lang.com/
4. http://gulpjs.com/
5. https://bower.io/
6. https://virtualenv.pypa.io/en/stable/
7. https://virtualenvwrapper.readthedocs.io/en/latest/
8. https://django-extensions.readthedocs.io/en/latest/graph_models.html
9. https://www.djangoproject.com/
10. https://www.postgresql.org/
11. https://www.digitalocean.com/community/tutorials/how-to-install-and-use-postgresql-on-ubuntu-16-04
12. https://el.wikipedia.org/wiki/%CE%A0%CE%BF%CE%BB%CF%85%CE%BC%CE%BF%CF%81%CF%86%CE%B9%CF%83%CE%BC%CF%8C%CF%82_(%CF%85%CF%80%CE%BF%CE%BB%CE%BF%CE%B3%CE%B9%CF%83%CF%84%CE%AD%CF%82)
13. Tango with Django- by Leif Azzopardi and David Maxwell
14. Learn python the hard way- Zed A. Shaw